The Next Wave

Vol. 25 | No. 1 | 2024

The National Security Agency's review of emerging technologies

ISSN 2640-1789 (Print) | ISSN 2640-1797 (Online)

CYBER ANALYTICS RESEARCH

[Photo credit: Composite iStock.com/gremlin]

THE Next Wave
The National Security Agency's review of emerging technologies

GUEST Editors' column

Robert J. Runser & Joe McCloskey

The current state of the art in securing cyber systems is best summed up as: *zero trust, always verify*. Over the last decade, significant technical strides have been made in architecting cybersecurity into the foundations of today's systems. Zero trust is one of these core security principles. Network administrators assume compromise each and every time a host or network device opens a connection to a critical resource, accesses a privileged account, or downloads and executes software. These network- and host-based activities are continuously analyzed by a zero-trust decision engine to make risk-informed decisions that prioritize protecting the network. A large enterprise, however, may log thousands of requests for host and network resources every minute, creating ever greater volumes of cyber network and host data that must be analyzed for malicious behavior. While zero trust has successfully encouraged network administrators to aggressively monitor every device on the system, the most difficult challenge is to determine the type and frequency of analytics to apply to this data to maintain security over the operating lifetime of the network.

This issue of *The Next Wave* explores research by NSA and our collaborators for improving the cybersecurity of complex systems through the development and deployment of cyber analytics that can operate on heterogeneous data that is collected from sensors deployed throughout the network. Researchers apply advanced concepts developed from machine learning (ML), topology, and graph theory to solve important cybersecurity problems such as detecting network anomalies, discovering new malware, and detecting the behavior of compromised devices.

We open the issue with a joint paper with our collaborators at the Pacific Northwest National Laboratory. As the article ''Stepping Out of Flatland" illustrates, modern data science and ML are constantly providing new methods to analyze cyber data. The authors provide a novel and sophisticated framework to analyze behavior in cyber network data using the theory of hypergraphs and concepts from the growing field of topological data analysis. The mathematical richness of a hypergraph allows the complex relationships in cyber log data to be captured and modeled effectively with topological features. The mathematical structure inherent in a hypergraph then allows these topological signatures to be used to analyze behavior and detect attacks. This approach has the advantage of overcoming the lack of interpretability of conventional methods.

Cybersecurity anomaly detection is a growing area of concern; the detection of malicious activity is of paramount importance in securing a network. One standard method, which relies on known vulnerabilities, is to use some form of signature-based method. Instead, in the article ''Cybersecurity Anomaly Detection Using Graph Vertex Degree Assortativity," the goal of the author is to analyze internal NetFlow data and identify subtle changes that may indicate malicious activity. To do this, network flow traffic is encoded as a directed graph called a traffic dispersion graph. Then one particular graph measure of connectivity, vertex degree assortativity, detects anomalous activity for different Internet protocols (IPs). Follow-on time series analysis analyzes and resolves IP anomalies that are uncovered. These techniques also take into account the constant change inherent in network flow and can be used to develop a semi-automated workflow for monitoring an enterprise network.

The article "A Federated Machine Learning Paradigm for Scalable Cyber Threat Detection" is an excellent example of the application of artificial intelligence (AI) and ML to the detection of anomalous behavior by intrusion detection systems in near real time. They address the challenges that arise because of the incompatibility between the need for a detailed view of network state provided by multiple sensors (i.e., local servers at the network's edge) and the constraints imposed with moving large amounts of data to a centralized repository for model training (i.e., the central server). To do this, they use federated learning; models are trained in a distributed environment on the local servers under the management of a central server. Feedback is aggregated and models are either updated for deployment or additional local training is requested by the central server. Notably, the raw data is stored locally and not transferred to the central server. The authors use unsupervised learning methods and neural net architectures as their main tool for detecting malicious or anomalous cyber activity.

We devote the rest of the issue to cyber analytic tools that have been developed by researchers that are operating on real-world data such as network packet captures and binary files. Authors from NSA's Laboratory for Advanced Cybersecurity Research collaborated with George Mason University to tackle the timely problem of detecting compromised Internet of Things (IoT) devices by leveraging existing ML image processing capabilities to analyze IoT wireless protocols for malicious behavior. The system provides both situational awareness and anomaly detection capabilities by providing alerts in an easy-to-use, interactive dashboard.

The next article, "Malware Bytes" explores the effectiveness of looking at the bytes of malware files using ML techniques on byte N-grams. This work, which leverages over a decade of research at the Laboratory for Physical Sciences, has been shown to detect previously unseen malware and has inspired new ways to improve the computing architecture used for training these algorithms and scanning the "fire hose of files" entering an enterprise network every day.

The final feature article reviews recent work at the Laboratory for Telecommunication Sciences to develop a new analytic tool to improve binary file analysis. The new tool, YELLOWTIGER, extracts features from binaries to help analysts identify similar code structures that may indicate malicious behavior, identify authorship, and predict provenance.

We thank the authors for their continuing contributions to the important field of cyber analytics research—an essential component in today's cybersecurity architecture. We also extend our thanks to Jessica for the outstanding editing, layout, and artwork throughout this issue.

Robert J. Runser
Technical Director
Research Directorate, NSA

Joe McCloskey
Technical Director of Mathematics Research
Research Directorate, NSA

Contents

The Next Wave is published to disseminate technical advancements and research activities in telecommunications and information technologies. Mentions of company names or commercial products do not imply endorsement by the US Government. The views and opinions expressed herein are those of the authors and do not necessarily reflect those of the NSA/CSS.

This publication is available online at http://www.nsa.gov/thenextwave. For more information, please contact jsmarx@uwe.nsa.gov.

ISSN 2640-1789 (Print)
ISSN 2640-1797 (Online)

Vol. 25 | No. 1 | 2024

STEPPING OUT OF FLATLAND: Discovering Behavior Patterns as Topological Structures in Cyber Hypergraphs

Helen Jenne, Sinan G. Aksoy, Daniel M. Best, Alyson Bittner, Gregory Henselman-Petrusek, Cliff Joslyn, Bill Kay, Audun Myers, Garret Seppala, Jackson Warley, Stephen J. Young, Emilie Purvine

Data breaches and ransomware attacks occur so often that they have become part of our daily news cycle. Last year, 1,802 data compromises affected 422 million people [1]. In a 2022 op-ed co-written by the Cybersecurity and Infrastructure Security Agency Director and National Cyber Director, they described the omnipresent threat of cyberattacks as “the new normal,” writing that in the modern landscape of complex cyber threats, “our shields will likely be up for the foreseeable future” [2]. This is due to a myriad of factors, including the increasing number of Internet-of-Things devices, shift to remote work during the pandemic, and advancement in adversarial techniques—all of which contribute to the increase in both the complexity of data captured and the challenge of protecting our networks. At the same time, cyber research has made strides, leveraging advances in machine learning and natural language processing to focus on identifying sophisticated attacks that are known to evade conventional measures. While successful, the shortcomings of these methods, particularly the lack of interpretability, are inherent and difficult to overcome. Consequently, there is an ever-increasing need to develop new tools for analyzing cyber data to enable more effective attack detection.

In this article, we present a novel framework based in the theory of hypergraphs and topology to understand data from cyber networks through *topological signatures*, which are both flexible and can be traced back to the log data. While our approach’s mathematical grounding requires some technical development, this pays off in interpretability, which we will demonstrate with concrete examples in a large-scale cyber network dataset. These examples are an introduction to the broader possibilities that lie ahead; our goal is to demonstrate the value of applying methods from the burgeoning fields of hypernetwork science and applied topology to understand relationships among behaviors in cyber data.

[Photo credit: iStock.com/krulUA]

From Flatland to Cyberland

In US and NATO military doctrine, cyberspace is often considered a domain, alongside land, sea, air, and space [3, 4]. But unlike these domains, we cannot travel into cyberspace and physically touch the system processes or digital information. Because of the lack of physical constraints in cyber, many have observed that thinking of it as a domain could be a hindrance by putting unnecessarily restrictive limits on what behaviors are possible [4, 5, 6]. In a recent article [6], Pierre Trepagnier of Massachusetts Institute of Technology Lincoln Laboratory contended that instead we should think of cyberspace as another dimension. He presented his argument using the classic novella *Flatland*, by Edwin A. Abbott, which takes place in a two-dimensional world inhabited by geometric figures. In the story, the narrator, a square, is visited by a sphere from three-dimensional Spaceland, who appears to be a circle as it passes through Flatland. Despite the sphere's best efforts, the square cannot comprehend three-dimensional space until the sphere takes him there. In his essay, Trepagnier identifies numerous parallels between our challenges to understand cyberspace and the square's inability to understand Spaceland. He writes,

> In our physical world, our senses have evolved to perceive threats directly. But we cannot perceive packets; our perception of cyber is entirely synthetic, through sensors which we place out in Cyberland and whose outputs we route back into [our world].

To elaborate on Trepagnier's description, sensors in Cyberland collect timestamped data called logs. Log data are generated from every device and application in the network, including routers, firewalls, workstations, and servers. Logs consist of digital observations, which vary in terms of the granularity, scope, and modality of information captured. For example, host logs are records of everything that happens at a system level, where a host is an individual computer, server, or other connected device. These records include login attempts, file creation and deletion, registry edits, errors, warnings, and other processes. Network logs offer a higher level perspective; they record information that crosses the boundaries of individual hosts like the flow of digital packets between hosts, and thus offer valuable insights into network-wide patterns. Gaining a complete picture of the current cyber landscape requires synthesizing information from host logs, network logs, and other data sources. Each stream of logs provides incomplete information, but they are linked through the imperceptible dimension of Cyberland. The challenge is modeling these data in such a way that recovers the unseen complexity and allows cyber analysts to take action based on the understanding they gain from the models.

OpTC data: Introduction and context

Before describing our approach to modeling cyber data, we introduce the dataset we will use throughout this article: the Operationally Transparent Cyber (OpTC) dataset [7]. The OpTC dataset was released by the Defense Advanced Research Projects Agency (DARPA) Transparent Computing program to enable research that enhances understanding of and defense against advanced persistent threats (APTs) at scale. An attack by an APT is an extended cyberattack (low and slow) that, for example, aims to steal highly sensitive data or to impose the will of the malicious actor. It involves multiple stages, including initial infiltration, expanding access, using that access to gain administrative credentials, moving laterally to other servers or workstations, and finally exfiltrating data.

The OpTC dataset contains over 17 billion events generated from a simulated network consisting of approximately 500 hosts, scripted to mimic daily user activities (e.g., downloading files from emails, editing files, and browsing the Internet), along with three days of annotated red team activity representing APT scenarios. The OpTC dataset brings both network and host logs together in a common format with common metadata fields, allowing one to make connections between system processes and other individual log events. We show an excerpt of the logs in table 1 to illustrate some of the detail provided in the dataset. These are a particular type of network logs, called network flow logs, that summarize host-to-host communications, such as a computer accessing a website or a user opening a remote desktop connection to access another computer. In the OpTC data, a single network flow log contains the start time of the flow (timestamp), the length of the flow (duration), the originating computer system [source Internet Protocol (IP) address], the recipient (destination IP address), the interface used to send and receive communication (source and destination port), and the method of sending data (protocol), in addition to other metadata such as the path to the program that started the event (image path) and which entity

TABLE 1. Excerpt of network flow logs of OpTC data[a]

time	action-object	host	principal	pid	source IP	dest IP	dest port	protocol	image path
13:51:28	START-FLOW	SysClient0501	bantonio	2956	142.20.57.246	142.20.61.189	3389	TCP	mstsc.exe
13:52: 08	MESSAGE-FLOW	SysClient0974	sbobertz	3768	142.20.59.207	153.129.45.5	80	TCP	firefox.exe
13:54:36	MESSAGE-FLOW	SysClient0974	sysadmin	3636	142.20.59.207	142.20.56.6	3389	TCP	mstsc.exe
13:55:40	START-FLOW	SysClient0811	rsantilli	5712	142.20.59.44	142.20.61.130	135	TCP	python.exe

is performing the action (principal). The richness of the data, documentation of red team attacks in the ground truth, and subsequent in-depth analysis of the network and host events [8] has led to the use of OpTC in a variety of research programs for cybersecurity threat detection [9, 10, 11, 12, 13].

Hypergraphs as models of cyber network data

Activity in the OpTC data (and cyber log data in general) is characterized by many complex relationships among groups of items recorded in the logs. For example, groups of ports can be related by virtue of the processes that use them; or groups of IP addresses can be related based on the protocols they employ for their various communications. To model and analyze the relationships present in complex data like the OpTC data, we turn to hypergraphs. As mathematical models of data with group-wise relationships, hypergraphs have provided great benefit in recent years to researchers across a variety of domains [14, 15, 16, 17, 18, 19].

A hypergraph consists of a set of vertices, representing individual entities, along with a collection of hyperedges, where each hyperedge is a subset of the vertices of any size and represents some joint property among the vertices it contains. For example, the relationship between users and destination ports can be naturally structured as a hypergraph where the users are vertices, and each destination port hyperedge contains the users that use that port, as shown in figure 1 (left and right). In general, when data come as individual records and each record contains the same field names, one can choose two fields (e.g., source IP address and destination port in network flow logs, or command line and user in host logs) and model the relationship among the items that fill those fields using a hypergraph.

As the name suggests, hypergraphs are generalizations of graphs, which model pairwise relationships by restricting hyperedges (called “edges” in this context) to be size two. In cyber data, graphs have been used to model communications between IP addresses, relationships between users, and

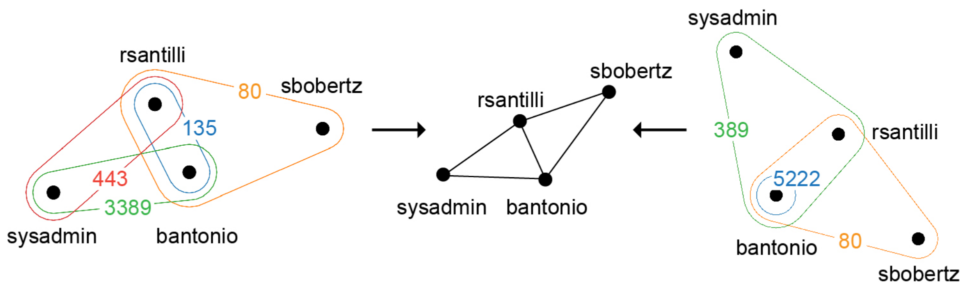


FIGURE 1. These two user–port hypergraphs (left and right) have the same graph representation (center). Both hypergraphs capture user behavior information through the hyperedge containment and intersection patterns that the graph cannot.

a. These are only a fraction of the keys (columns) recorded in the data. There are common keys shared among all log types, like action-object, host, principal, process ID, and others, but also fields unique to each log type.

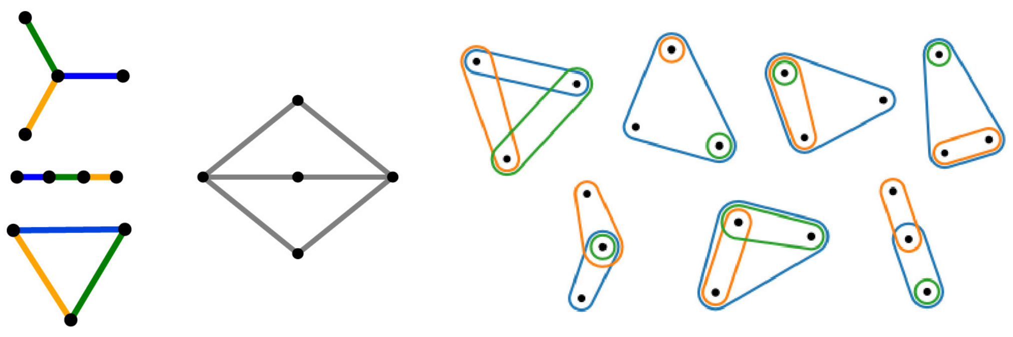


FIGURE 2. The first column shows three-edge graph motifs; the second column shows an example of a connector motif, and the third column shows seven of the 27 three-hyperedge hypergraph motifs.

similarities among malware [10, 20, 21, 22, 23, 24, 25]. But graphs should be used with caution in data with complex relationships; as we learned from Flatland, appearances when projecting down from higher dimensional spaces can be deceiving. For example, the aforementioned user–port relationship with a hypergraph can also be represented as a graph where two user vertices are linked by an edge when they perform actions using the same port. However, modeling the data with a graph results in information loss: the same user–user graph can correspond to many user–port hypergraphs, as shown in figure 1. As the examples in this figure illustrate, hypergraphs can capture complexities that graphs cannot. While two edges in a graph can only interact by sharing one vertex, two hyperedges in a hypergraph can interact in many more ways. Hyperedges can intersect in any number of vertices, or one hyperedge can be fully contained within another. If we think of hyperedges as representing groups of vertices that share a behavior, the hypergraph perspective allows us to model how behaviors are linked or related by virtue of the vertices which display that behavior. The question then is how to use these models to gain understanding of the data and the behaviors therein.

A recent and active area of research to study hypergraph models of data is "hypernetwork science" [26], taking inspiration from the well-established field of "network science,"[b] which is the analysis of graph models of real-world data. Often in hypernetwork science, methods that were developed to analyze graph models are extended and generalized to apply to hypergraphs; but in many cases, this is not a trivial task. One such graph method that has proven valuable across a wide range of fields is the study of how small, connected graph substructures (also known as "motifs") appear in large graphs. A graph motif is nothing more than a small connectivity pattern or signature. Some of the simplest graph motifs include the triangle, square, *n*-star, and *n*-path. All 3-edge motifs—the triangle, 3-star, and 3-path—are illustrated in figure 2 (left). In cyber graphs, the "connector," where multiple vertices all connect to the same pair of vertices (an example with three central vertices is pictured in figure 2), and *n*-star motifs are common and indicative of certain kinds of network behaviors and configurations [27].

In order to extend motifs to hypergraphs, with the straightforward generalization being small, connected subhypergraphs, we look to recent work that defines 27 hypergraph motifs with three hyperedges [28] and develops algorithms to count them. But the authors of that work note that generalizing their hypergraph motifs to four and five hyperedges results in a combinatorial explosion: there are 1,853 four-hyperedge hypergraph motifs (compared to 5 four-edge graph motifs) and over 18.6 million five-hyperedge hypergraph motifs (compared to 12 five-edge graph motifs). To illustrate just some of the complexities of hypergraph motifs, we again refer to figure 2 (right) where seven of the 27 three-hyperedge hypergraph motifs are shown, those that can occur with just three vertices. As we will see in the OpTC data, these complexities, in particular the

b. The terms "graph" and "network" are often used synonymously, with "network" usually connoting when a graph is modeling real-world data. However, from here on we will use the term "graph" to avoid confusion with the cyber use of the word "network."

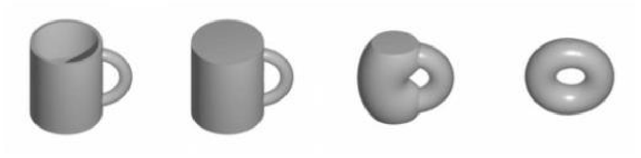


FIGURE 3. A donut and a coffee cup are topologically the same because the coffee cup can be continuously deformed into a donut. [Image credit: Lucas Vieira, Public domain, via Wikimedia commons]

presence of single-vertex hyperedges and hyperedge containment, are important for capturing different behavior patterns. Because of their combinatorial complexity, for hypergraph motifs to be useful in characterizing these patterns, they must be studied as a small number of groups of motifs where each group contains motifs of "similar enough" structure. While one could imagine many ways to define similar enough structure, resulting in different groupings of motifs, one way is to turn to the mathematical field of topology, which provides techniques for summarizing structure in a flexible and interpretable way. This forms the basis of our framework; we define topological signatures in hypergraphs by grouping motifs that are topologically similar.

The topological perspective

Topology studies different shapes with a flexible notion of what it means for two shapes to be "the same," where two shapes are considered the same if they can be continuously deformed (via stretching, twisting, bending, but not breaking, tearing, or puncturing) into one another. Figure 3 shows a classic example of a continuous deformation between a coffee cup and a donut. This concept is made rigorous by studying what are called topological invariants. A topological invariant is a mathematical quantity that identifies certain properties of shapes and is guaranteed to be equal for two shapes that are the same topologically. For the last 20 years, topological invariants have been used as a tool to make sense of high-dimensional data in a field called topological data analysis. One of the most studied topological invariants in the data science community is called homology, which informally is a method of counting holes in a shape. The number of connected components (zero-dimensional holes), loops (one-dimensional holes), voids (two-dimensional holes, like the inside of a basketball), and higher dimensional analogs are topological invariants that

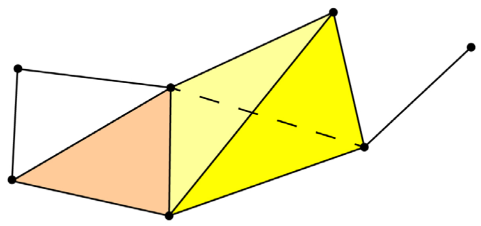


FIGURE 4. This topological object is composed of points, lines, solid triangles (orange), and solid tetrahedra (yellow).

conveniently encode multidimensional structures in a way that has proven useful in many applications and for different data types [29, 30].

Core to the notion of homology is the idea of a continuous deformation. But it is not immediately clear what that means in the case of a hypergraph. While we can draw a hypergraph on a page, it is not something we can hold in our hands and twist or bend. To leverage the topological invariant of homology to create topological signatures in the context of hypergraphs, we need a method to translate hypergraphs into topological objects that we can deform. Example building blocks of such tangible topological objects, in dimensions we can perceive, are points (zero-dimensional), lines (one-dimensional), solid triangles (two-dimensional), and solid tetrahedra (three-dimensional); these building blocks are shown connected together to form an example topological object in figure 4. It will be important later to know that these building blocks also include their lower dimensional boundaries. For example, a triangle includes its three edges and three vertices; a tetrahedron includes its four triangular faces, six edges, and four vertices. Higher dimensional analogs exist, and even though we cannot hold them, the theory for computing their homology extends seamlessly. Translating from a hypergraph to one of these topological objects can be done in many ways, and the topological holes of each one will tell us something different about the hypergraph and the data that was used to construct it.

Translating hypergraphs into the language of topology

Our research team is exploring various methods to translate hypergraphs into topological objects and what insight the homology of these objects provides in the context of cyber data. As we will see, after finding a hole in a topological translation of a hypergraph, we can identify the hypergraph motif that gives rise to the hole. When multiple hypergraph motifs give rise to the same dimension of hole, we group them as motifs that are topologically similar through the lens of that translation. This group of motifs is then a topological signature.

We have identified two hypergraph translation methods of particular interest, which result in two types of groupings of motifs into topological signatures, but there are others that could also lead to interesting insights into cyber datasets [31, 32]. We briefly mention the first and provide a reference to work where we show its use in cyber. We then go into more detail in the second construction, including grounding it by interpreting the holes in the context of benign and adversarial activity in the OpTC data.

Perhaps the most straightforward way to create a topological object from a hypergraph is by replacing each hyperedge of size *k* with a solid (*k*-1)-dimensional building block. We will refer to this as the closure of the hypergraph. In figure 5, we show three example hypergraphs (first column) with their closures (second column). Notice that two of the hypergraphs in this figure have the same closure, which has a one-dimensional hole. Therefore, these hypergraphs are topologically similar motifs through the lens of the hypergraph closure. This is because forming the closure of a hypergraph is akin to adding every subset of every hyperedge to the hypergraph, whether or not it was there to begin with. When a hyperedge of size four is present, for instance, we replace it with the solid tetrahedron that also includes all the 3-way triangular faces, 2-way edges, and singleton vertices. Recently our team studied how motifs giving rise to holes in the closure of hypergraphs that capture the relationship between source IP address and executable (e.g., python.exe) in small time windows in the OpTC data change over time [33]. In that work, unusual patterns of topological change correlated with some instances of malicious network behavior recorded in the ground truth.

Motifs found using the closure of a hypergraph identify what we might consider as intrinsic cycles, those that you can see if you just look at a drawing of the hypergraph. This is certainly a valuable perspective but forming the closure only provides one way of studying how behaviors, captured by hyperedges, interact. Another approach comes from the earlier observation that a hyperedge can be interpreted as a group of vertices sharing a behavior. From this perspective, we posit that hyperedge containment relations are important since they mean that a group of vertices that share one behavior (the smaller hyperedge) also fully share another behavior (the larger hyperedge). This position leads us to introduce our second method to translate a hypergraph into a topological object that we call the nesting object of the hypergraph. To put it concisely, the nesting object includes a vertex for each hyperedge in the hypergraph and a (*k*-1)-dimensional building block for every sequence of *k* hyperedges where one is a subset of the next. To explain how this construction relates to our assertion that hyperedge containments are important, we build the nesting object in a two-step process.

In the first step of the translation, we create the hyperedge containment graph (HCG) of a hypergraph that has one vertex for each hyperedge, and an edge in the HCG represents a hyperedge containment relationship in the hypergraph. Concretely we add an edge in the HCG between the vertices corresponding to two hyperedges whenever one of the hyperedges is contained in the other. The third column of figure 5 shows the HCG for the same three example hypergraphs. Since any graph, and in particular our HCG, consists of points (vertices) and line segments (edges), it is already a topological object in that sense, and thus can contain one-dimensional holes (loops). But it cannot contain any higher dimensional holes. Moreover, the one-dimensional holes in the HCG can be generated from different types of hypergraph substructures. Specifically, we define a nest of hyperedges as a sequence of hyperedges, each one a subset of the next. In the case of three hyperedges, this is when one hyperedge is completely contained in another hyperedge, which is itself completely contained in a third hyperedge. Then in the HCG, there is a 3-cycle between those three hyperedges. These nests are interesting in their own right, but they are easy to identify in the HCG as the completely connected subgraphs; the machinery of homology is not needed to identify nests. Other holes in the HCG encode relationships among the nests, and finding those holes can point to more complex behavior patterns. To find these we need to look not at the nests themselves, but at the relationships between them. This is the motivation behind the second step of our process, to enrich the HCG with higher dimensional structure to distinguish between the nests and the relationships among the nests.

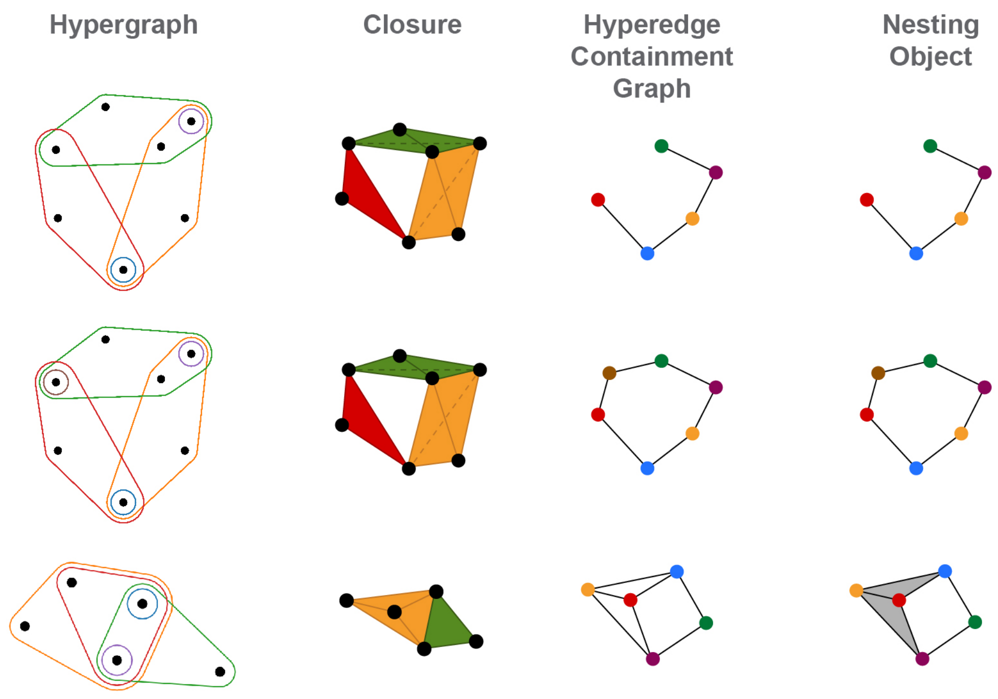


FIGURE 5. In the first column are three small hypergraphs with their closure (second column), hyperedge containment graph (third column), and nesting object (fourth column). In the closure, the topological building blocks (triangles and tetrahedra) are colored corresponding to the hyperedges that give rise to them. In the hyperedge containment graph and nesting object, the vertices are colored the same as their corresponding hyperedges.

For homology to ignore cycles created by these nests and find other complex containment patterns among the nests we must "fill in" the one-dimensional holes they create. Specifically, we replace every cycle of length 3 in the HCG with a solid triangle, and every set of 4 vertices that are completely connected (known as a *4-clique*, and representing a nested sequence of 4 hyperedges) with a solid tetrahedron. We do this in general for all *k*-cliques (set of *k* vertices that are all pairwise connected) in the HCG, replacing them with (*k*-1)-dimensional analogs of tetrahedra. The result of this procedure is the nesting object.[c] We noted above that homology of the closure identifies "intrinsic" holes in the hypergraph. In the case of the nesting object, we understand that one-dimensional holes correspond loosely to cycles in which hyperedges are fully contained within intersections of other hyperedges (although the exact structure that is identified through homology of the nesting complex is more nuanced, especially in higher dimensions). We can see this difference between holes in the closure and nesting object illustrated in figure 5, where the rightmost column shows the nesting object of our three example hypergraphs. What is interesting is that, while the top two hypergraphs had the same closure, it is now the bottom two that have topologically similar nesting objects. Both have a one-dimensional hole. The cycles are different lengths, 4 and 6, but since the overall pattern of pairs of intersecting hyperedges containing a hyperedge within their intersection is the same, these two small hypergraph motifs that give rise to the one-dimensional holes are grouped as topologically similar. The top hypergraph in the figure is missing one of the hyperedges in the intersection of two larger hyperedges which breaks the cycle and so it is not topologically similar through the lens of the nesting object.

We now have all the mathematical background needed to explore the OpTC data through the nesting object of a specific hypergraph construction. We will see how groups of hypergraph motifs that are topologically similar arise from similar behaviors, many of which correspond to adversarial activity.

c. The technical term is *restricted barycentric subdivision*. If you are interested in learning more about the technical details see [32, 34, 35].

Topological signatures in the OpTC data

In the remainder of this article, we delve into examples from the OpTC data where we look for structure in the form of topological signatures. These examples show how the homology of the nesting objects of hypergraphs provided insight into interpretable patterns of activity that were often tied to malicious actions. As previously described, due to the comprehensive nature of the OpTC dataset there are many options for hypergraph constructions. Topological holes that arise from different choices of vertices and hyperedges could lead to the discovery of distinct but equally significant behaviors. For this investigation we focus on hypergraphs where the vertices represent (source IP, host) pairs (as these identify machines) and the hyperedges represent (destination IP, destination port) pairs to model the way that machines interact with other systems on the network (here, and for the remainder of the article, we abbreviate "IP address" to "IP"). We restrict each hypergraph to a 10-minute time window of data. As a reminder, this means that a vertex is in a hyperedge if there is network flow log with that (source IP, host, destination IP, destination port) tuple during the time window represented by the hypergraph. To simplify exposition, we will refer to the vertices as "machines" rather than their (source IP, host) pair. We will see that significant aspects of the red team activity, including communication with the command and control (C2) server and lateral movement, manifest as nontrivial homology of this hypergraph's nesting object.

Adversarial behavior as holes

Each of the three days of red team activity in the OpTC data had a different APT attack campaign involving different hosts and users and so can be studied independently; here, we focus on the second day. At a high level, the second day represents a custom PowerShell Empire scenario. PowerShell Empire is a toolkit designed to streamline the phases of an APT attack following initial infiltration. It enables attackers to escalate privileges and gather intelligence, while evading detection. For example, by using PowerShell Empire an attacker can run scripts and modules in memory to stay more easily hidden. Specifically, it provides C2 capabilities that facilitate communication between the attacker and the compromised hosts by generating a listener that is protocol based (often HTTP, HTTPS, or DNS), so the communication looks like normal network traffic.

At the beginning of the second day of the red team activity, the users *bantonio* and *rsantilli* were compromised when they opened malicious attachments from phishing emails on their workstations. As soon as they were compromised, bantonio's and rsantilli's machines began to communicate regularly with the two C2 servers, whose IPs we will refer to as C2A and C2B. In the logs, this means that there are records containing bantonio's machine, with the destination IP C2A using the port for HTTPS traffic. Similarly, there are frequent records containing rsantilli's machine, with the destination IP C2B using the HTTP port. In the hypergraphs, this manifests as a hyperedge labeled (C2A, HTTPS) containing a single vertex corresponding to bantonio's machine, and a (C2B, HTTP) hyperedge containing a single vertex corresponding to rsantilli's machine. The former hyperedge appears in all hypergraphs overlapping with the time where bantonio's machine was active, 10:36 until 15:28.[d] The latter hyperedge appears in all hypergraphs overlapping with the time interval 10:40 to 13:25, when rsantilli's machine was active. Later in the afternoon, across different time windows, the single-vertex (C2B, HTTP) hyperedge instead contains vertices corresponding to other machines, consistent with the lateral movement documented in the diary of red team activity.

These singleton hyperedges that correspond to the C2 communications do not, by themselves, create any topological holes in the nesting object. More hyperedges are needed to tie those behaviors together. In addition to the communications with C2A and C2B, rsantilli's and bantonio's machines also communicated with the same domain controller (DC)[e] across two different channels of communication (HTTP and HTTPS) shortly after they were compromised. Consequently, the large, complex hypergraph representing this time interval (figure 6, left) contains the motif shown in figure 6 (center). The nesting object of this motif is a cycle of length 4 shown in figure 6 (right), which is very similar to the bottom example in figure 5. This example illustrates a one-dimensional hole in the nesting object whose corresponding motif contains hyperedges corresponding to adversary behavior.

d. The port changes from HTTPS to SSH at 13:11 when bantonio starts a reverse SSH connection.

e. A DC is a machine responsible for managing network security requests. It provides a service to ensure that only users with authorization on a resource are allowed access.

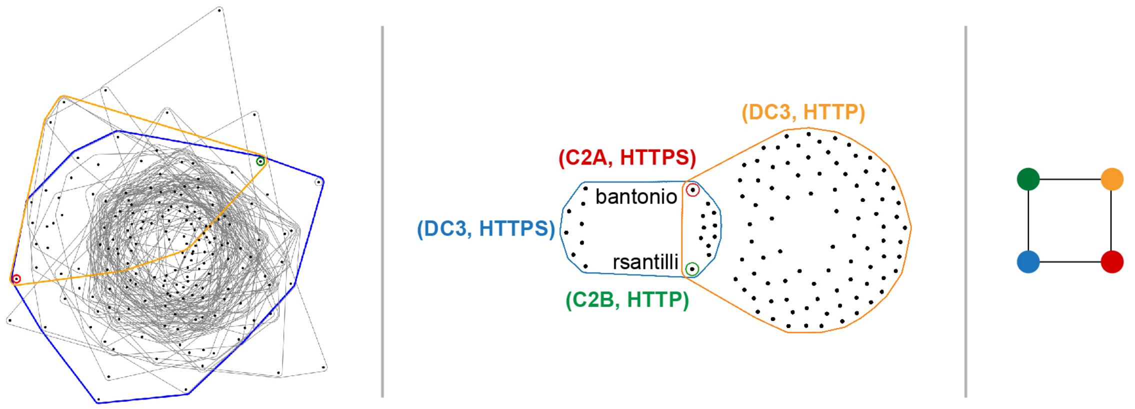


FIGURE 6. On the left is the full hypergraph corresponding to one 10-minute time window whose nesting object has a single one-dimensional hole. The motif representing that hole is shown in the center (hyperedges are also highlighted in the full hypergraph), and the nesting object is shown right.

Interestingly, more than half of the motifs we found in hypergraphs within the time period of malicious activity had this motif of two single-vertex hyperedges in the intersection of two large hyperedges (38 out of 64). Figure 7 shows two more motifs (that created holes in the nesting complexes of hypergraphs representing 11:00-11:10 and 13:45-13:55, respectively) that, at first glance, appear identical to the one in figure 6, but there are some subtle differences in what the hyperedges represent. In the left motif, the single-vertex hyperedges again contain vertices representing bantonio's and rsantilli's machines, and the blue and orange hyperedges show that these machines were communicating with a common DC, but the red hyperedge containing bantonio's machine represents not one destination IP and port hyperedge but 335 of them. Close inspection of the logs shows that these hyperedges represent network flow records that are a downstream effect of the attacker's attempt to elevate the permission level of their account on bantonio's machine using PowerShell Empire. The motif on the right of figure 7 represents the attacker moving to different hosts using remote desktop sessions. Specifically, the red hyperedge represents when the attacker used a remote desktop session to move to host 974, and the green hyperedge represents when the attacker moved from host 974 to host 005 a few minutes later. These differences illustrate that this common nesting object structure is not just representing one adversary tactic; *rather, this structure is consistent with the way that those behaviors interconnect, not with the behaviors themselves.*

Even when the motifs do not have this structure of two single-vertex hyperedges in the intersection of two large hyperedges, they still often demonstrated

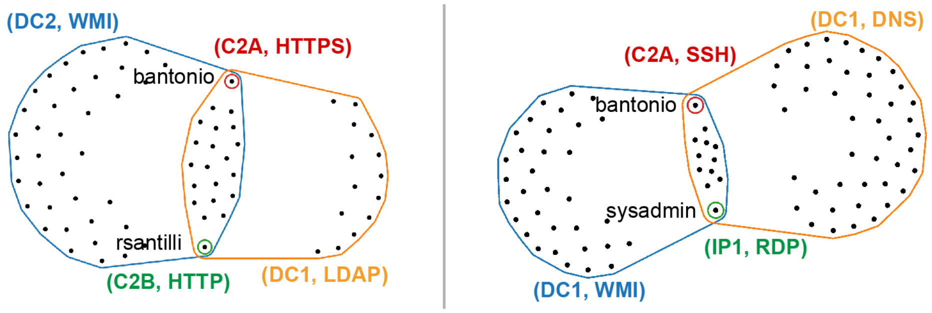


FIGURE 7. These two additional sub-hypergraphs of 10-minute windows have cycles in their nesting complexes. The structure is the same as in figure 6 but the behaviors that the hyperedges represent are slightly different.

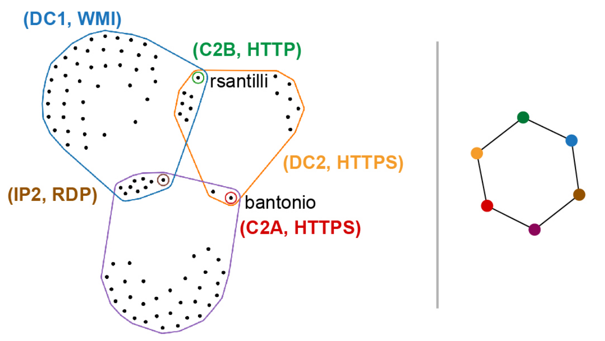


FIGURE 8. This sub-hypergraph of a 10-minute window (left) was identified by our algorithm to have a cycle in its nesting object (right). This is a 6-cycle as opposed to a 4-cycle in figures 7 and 8 but was found as a result of our search for one-dimensional holes.

connections to adversary behavior. For example, figure 8 shows a motif that has the same hyperedge containment relationships as the middle hypergraph in figure 5 and, therefore, also has a cycle of length 6 as its nesting object. The hyperedges in the motif represent bantonio's and rsantilli's machines talking to their C2 servers (the red hyperedges and green hyperedges, respectively), along with a benign remote desktop session from another machine (the brown hyperedge) and communication with DCs (the three large hyperedges). The commonalities between this and the examples in figures 7 and 8 illustrate the need for flexibility of motifs and topological signatures, not just in theory but when working with real data. If we had just been searching for motifs that looked exactly like the examples in figures 7 and 8, we would have missed this.

Common interpretations of motifs

In traditional cyber research, signature-based methods search for patterns that match known adversary tactics, techniques, and procedures and raise an alert when a match is found. In contrast, our framework defining topological signatures is designed to discover more flexible patterns that occur in higher dimensional structures of cyber data, and what those patterns mean. But also of great value are methods in exploratory data analysis (EDA), which identify significant patterns from the bottom up. In this exposition we built our examples to clarify the connection to the ground truth adversary behavior. But in practice we made these discoveries through EDA by applying our topology software [36] to the full hypergraph from each time window to detect any instances of homology in its nesting object. We did not know what kinds of behaviors the motifs it found would represent; by inspecting which machines and (destination IP, destination port) pairs the vertices and edges represented, we discovered that many of the motifs were linked to the C2 communication from compromised machines. While not every motif could be linked to malicious activity (although many could, discussed below), a key finding of our work is that most of the motifs were interpretable, in that their hyperedges could be inspected and understood as linked behaviors. Moreover, there were commonalities in the interpretations.

This discovery is summarized in figure 9, which gives a timeline showing which elements of the red team's activity commonly show up across the motifs. The portion of the timeline where the red team was active is highlighted in red, and the gray cells indicate time windows for which the nesting object of the hypergraph had holes. Note that not all time windows had holes. If the hypergraph did have at least one hole, we show whether any of their corresponding motifs had features we have discussed. The appearance of bantonio's and rsantilli's machines in single-vertex hyperedges are indicated with red and blue dots, respectively; a green dot indicates when the single-vertex hyperedge contained the administrator on host 974; a yellow dot indicates when the single-vertex hyperedge represented a remote desktop protocol (RDP) session (benign or malicious); finally, we indicate when at least one of the large hyperedges of a motif represented communication with a DC with an orange outline. The last two characteristics are not alone indicative of malicious activity; for example, sometimes an RDP session is initiated by an adversary, but other times it is benign. However, we included these characteristics in the figure to emphasize that even when the motif was not linked to malicious activity, it was still interpretable most of the time. In fact, there are only two time windows that have a hole but none of these annotations.

It is remarkable that application of our framework—building a hypergraph, creating the nesting object, and computing the homology—discovers behavioral interactions joined together into similar patterns that we can interpret, even though there are differences in the finer details of the activity.

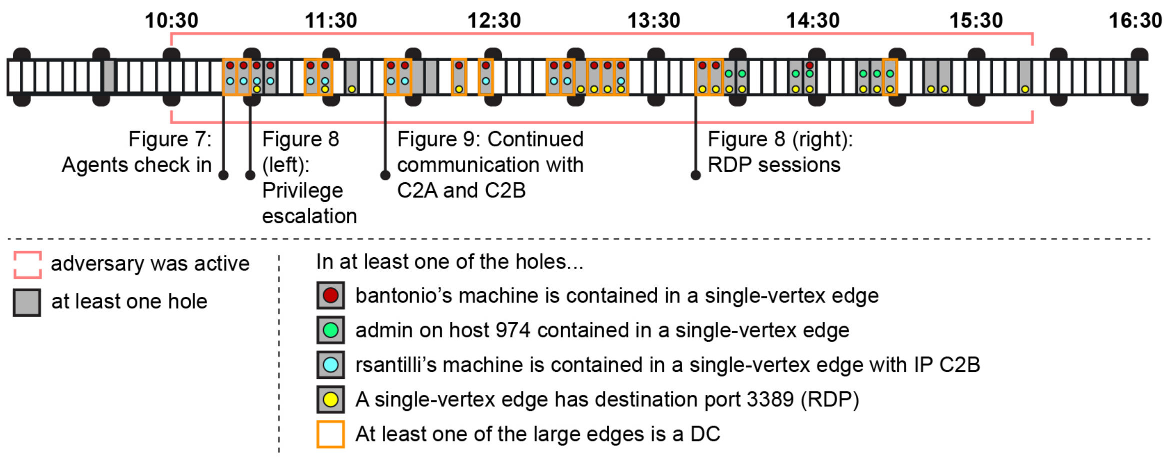

10:30
11:30
12:30
13:30
14:30
15:30
16:30
Figure 7: Agents check in
Figure 8 (left): Privilege escalation
Figure 9: Continued communication with C2A and C2B
Figure 8 (right): RDP sessions
adversary was active
at least one hole
In at least one of the holes...
bantonio's machine is contained in a single-vertex edge
admin on host 974 contained in a single-vertex edge
rsantilli's machine is contained in a single-vertex edge with IP C2B
A single-vertex edge has destination port 3389 (RDP)
At least one of the large edges is a DC

FIGURE 9. This annotated timeline displays our findings. We highlight (gray) which 10-minute windows have one-dimensional holes in their nesting objects. We additionally indicate when there is a one-dimensional hole in a window that contains elements of adversary activity or DC communications.

Evaluating holes as cyber alerts

Having established the success of our method at discovering and interpreting patterns, we look toward the challenge of turning topological signature identification into a cyber alert specifically designed to capture behavior that aligns with known adversary tactics. Although future work is needed to achieve that goal, in order to identify next steps we describe and study false positives and false negatives as they present in our current analysis.

False positives

If critical components of a motif (e.g., the single-vertex hyperedges included in larger hyperedges) could be linked to malicious behavior via process ID in the ground truth, we consider the motif to be malicious, a true positive. Of the 64 motifs in 31 time windows, 47 motifs (73.4 percent) were malicious. These 47 motifs occurred in 24 of the time windows, and when multiple motifs occurred in one time window, they were often very similar (for example, differing by a single hyperedge). The remainder of the motifs are false positives, but 13 of them were found to be related to benign RDP sessions. This leaves only 4 motifs (6.25 percent) that were not interpretable.

Another way we could quantify false positives is by counting motifs in hypergraphs constructed over a benign time period in the data with comparable levels of activity to the times of compromise. Unfortunately, the heterogeneity of the OpTC data makes this difficult. To approximate a benign time period, we used the same data but eliminated all network flow logs collected from compromised hosts before recomputing the homology. When we did this, we saw the number of vertices and hyperedges did not change much (figure 10, left), but the average number of holes per time window decreased from 1.0 to 0.43, and the majority of time windows had no homology (53 out of 64 had no homology, compared to 33 out of 64 with all hosts), which is shown in figure 10 (page 14, right). It may not seem surprising that the number of holes is smaller after removing some vertices and hyperedges. But notice that in figures 7, 8, and 9, there are many vertices (machines) that we could remove from the motifs without changing the fact that there is a one-dimensional hole in the nesting object. In these cases, it is the compromised machines (vertices) and the communications resulting from that compromise (hyperedges) that are crucial to the existence of the holes. In other words, it is *which* vertices and hyperedges we removed that is important.

False negatives

Our analysis of false negatives is more nuanced because of the choices built into our framework, namely the choice of hypergraph construction. To mirror the definition of a false positive, we could consider any adversary action that was not part of a

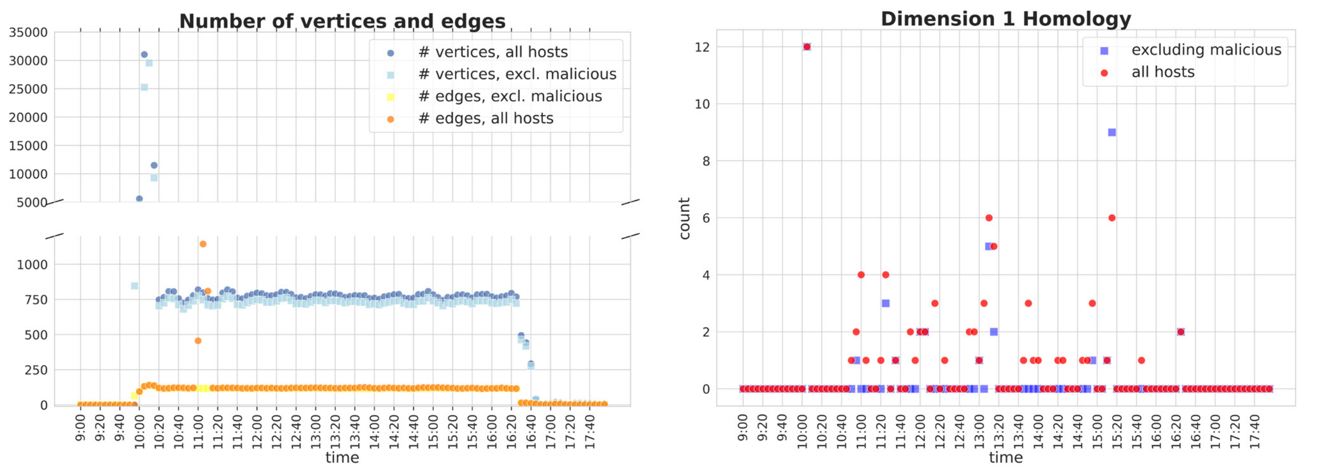


FIGURE 10. The number of vertices and hyperedges in hypergraphs with (source IP, host) vertices and (destination IP, destination port) hyperedges constructed from day 2 of the OpTC data are displayed in the scatter plot on the left, and the number of one-dimensional holes in these hypergraphs is displayed in the scatter plot on the right. To quantify homology unrelated to adversarial activity, we constructed hypergraphs both using data from all the hosts and a subset of the data that excludes the compromised hosts. Note that malicious activity occurs from 10:28 to 15:42.

motif as a false negative. However, there are a variety of different behaviors during an APT attack, some of which may primarily show up in host logs rather than network flow logs. The variety of adversary tactics mean that there will not likely be a single hypergraph construction that detects all malicious activity. While holes in the nesting object of the hypergraph construction we focused on seem to consistently identify similar patterns of behavior, more work is needed to discover hypergraph constructions that can be used to detect other adversarial behaviors in this dataset and others.

Additionally, we point out that when the adversary was active, the single-vertex (C2A, HTTPS) and (C2B, HTTP) hyperedges were present in each hypergraph. But in some time windows these hyperedges are not part of a motif giving rise to a hole. This is because our framework is designed to detect the linkage of behaviors in interesting ways, rather than single behaviors. Once adversary behavior is discovered in one time period as part of an instance of a topological signature, the individual components of the signature can be used to seed analyst queries for other instances of those behaviors.

Conclusion

The problem of developing new cyber tools that enable effective attack detection in a way that keeps up with adversary developments is challenging, in part because of the unknown complexity within the data. The framework we introduce here, identifying behavior linkage patterns through topological signatures, answers the challenge of stepping out of Flatland into the unknown dimension of Cyberland and points to a wide area of potential research. Our exploration of the OpTC data provides one example of its utility to make sense of this complex cyber data. Unlike traditional signature-based cyber tools, our methods did not set out to define patterns based on known adversary techniques, rather we modeled the complexity using hypergraphs and looked for structure via homology. It was very surprising and exciting that homology of the nesting objects took these hypergraphs with around 750 vertices and 100 hyperedges each and found small hypergraph motifs that were typically interpretable and often included adversary activity. There is still a need for more theory to understand the homology of topological translations of hypergraphs, interpretation to flesh out how topological signatures relate to user and adversary behaviors, and a bridge to be built between theoreticians and cyber analysts to ensure both sides understand and can help shape the perspective of the other. Our team will continue to explore these directions and we look forward to collaborations and partnerships as we continue our journey through Cyberland.

References

[1] Petrosyan A. Cyber crime: Number of compromises and impacted individuals in U.S. 2005-2022. *Statista.* 2023 Aug 29. Available at: https://www.statista.com/statistics/273550/data-breaches-recorded-in-the-united-states-by-number-of-breaches-and-records-exposed/.

[2] Easterly J, Inglis C. "'Shields Up': the new normal in cyberspace." *Cyberscoop.* 6 June 2022. Available at: https://cyberscoop.com/shields-up-easterly-inglis-op-ed/.

[3] Evans S. "Cyberspace is new domain for war: NATO." *Infosecurity Magazine.* 16 June 2016. Available at: https://www.infosecurity-magazine.com/news/cyberspace-is-new-domain-for-war/.

[4] Kreuzer M. "Cyberspace is an analogy, not a domain: Rethinking domains and layers of warfare for the information age." *The Bridge.* 2021 July 8. Available at: https://thestrategybridge.org/the-bridge/2021/7/8/cyberspace-is-an-analogy-not-a-domain-rethinking-domains-and-layers-of-warfare-for-the-information-age.

[5] Karas TH, Moore JH, Parrott LK. "Metaphors for cyber security." Sandia Technical Report SAND2008-5381. Sandia National Laboratory, NM: 2008. Available at: https://doi.org/10.2172/947345.

[6] Trepagnier P. "Cyber metaphors and cyber goals: Lessons from "Flatland." *Military Cyber Affairs.* 2019;4(1):2. doi: 10.5038/2378-0789.4.1.1045.

[7] DARPA. Operationally Transparent Cyber (OpTC) Data Release, 2019. Available at: https://github.com/FiveDirections/OpTC-data/.

[8] Anjum MM, Iqbal S, Hamelin B. "Analyzing the usefulness of the DARPA OpTC dataset in cyber threat detection research." In: *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies;* 2021; pp. 27–32. doi: 10.1145/3450569.3463573.

[9] Anjum MM, Iqbal S, Hamelin B. "ANUBIS: A provenance graph-based framework for advanced persistent threat detection." In: *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing;* 2022 Apr; pp. 1684–1693. doi: 10.1145/3477314.3507097.

[10] Bowman B. "Graph techniques for next generation cybersecurity." Doctoral dissertation, George Washington University; 2022.

[11] Cochrane T, Foster P, Chhabra V, Lemercier M, Lyons T, Salvi C. SK-Tree: A systematic malware detection algorithm on streaming trees via the signature kernel. In: *IEEE International Conference on Cyber Security and Resilience (CSR);* 2021 July; pp. 35–40. doi: 10.1109/csr51186.2021.9527933.

[12] Golczynski A, Emanuello JA. "End-to-end anomaly detection for identifying malicious cyber behavior through NLP-based log embeddings." 2021. ArXiv: 2108.12276.

[13] Mamun M, Shi K. "DeepTaskAPT: Insider apt detection using task-tree based deep learning." In: *IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom);* Oct 2021; pp. 693–700. doi: 10.1109/trustcom53373.2021.00102.

[14] Iacopini I, Petri G, Barrat A, Latora V. "Simplicial models of social contagion." *Nature Communications.* 2019;10:2485. doi: 10.1038/s41467-019-10431-6.

[15] Johnson J. *Hypernetworks in the Science of Complex Systems.* London: Imperial College Press; 2013. Available at: http://dx.doi.org/10.1142/9781860949739_0006.

[16] Klamt S, Haus U-U, Theis F. "Hypergraphs and cellular networks." *PLoS Computational Biology.* 2009;5(5):e1000385. Available at: https://doi.org/10.1371/journal.pcbi.1000385.

[17] Landry NW, Restrepo JG. "The effect of heterogeneity on hypergraph contagion models." *Chaos: An Interdisciplinary Journal of Nonlinear Science.* 2020;30(10). doi: 10.1063/5.0020034.

[18] Mann V, Venkatasubramanian V. "AI-driven hypergraph network of organic chemistry: Network statistics and applications in reaction classification." *Reaction Chemistry & Engineering.* 2023;8(3):619–635. doi: 10.1039/d2re00309k.

[19] Patania A, Petri G, Vaccarino F. Shape of collaborations. *EPJ Data Science.* 2017;6:18. doi: 10.1140/epjds/s13688-017-0114-8.

[20] Aksoy SG, Nowak KE, Purvine E, Young SJ. "Relative Hausdorff distance for network analysis." Applied Network Science. 2019;4:1-25. doi: 10.1007/s41109-019-0198-0.

[21] Aksoy SG, Purvine E, and Young SJ. "Directional Laplacian centrality for cyber situational awareness." *Digital Threats: Research and Practice (DTRAP).* 2021;2(4):1-28. doi: 10.1145/3450286.

[22] Bhattarai B. "Scalable graph pattern matching for cyber threat hunting." Doctoral dissertation, George Washington University; 2022.

[23] Chowdhury S, Khanzadeh M, Akula R, Zhang F, Zhang S, Medal H, Marufuzzaman M, Bian L. "Botnet detection using graph-based feature clustering." *Journal of Big Data.* 2017;4:1–23. doi: 10.1186/s40537-017-0074-7.

[24] Joslyn CA, Aksoy SG, Arendt D, Firoz J, Jenkins L, Praggastis B, Purvine E, Zalewski M. "Hypergraph analytics of domain name system relationships." In: In: Kamiński B, Prałat P, Szufel P, editors. *Algorithms and Models for the Web Graph.* WAW 2020. Lecture Notes in Computer Science, vol 12091. Springer, Cham; 2020. Available at: https://doi.org/10.1007/978-3-030-48478-1_1.

[25] Ramalingam D, Chinnaiah V. "Fake profile detection techniques in large-scale online social networks: A comprehensive review." *Computers & Electrical Engineering*. 2018;65:165–177. doi: 10.1016/j.compeleceng.2017.05.020.

[26] Aksoy SG, Joslyn C, Ortiz Marrero C, Praggastis B, Purvine E. "Hypernetwork science via high-order hypergraph walks." *EPJ Data Science*. 2020;9(1):16. doi: 10.1140/epjds/s13688-020-00231-0.

[27] Shi L, Liac Q, Sun X, Chen Y, and Lin C. "Scalable network traffic visualization using compressed graphs." In: *IEEE International Conference on Big Data;* 2013. pp. 606–612. IEEE. doi: 10.1109/bigdata.2013.6691629.

[28] Lee G, Ko J, Shin K. "Hypergraph motifs: Concepts, algorithms, and discoveries." 2020. ArXiv: 2003.01853.

[29] Carlsson G, Vejdemo-Johansson M. *Topological Data Analysis with Applications.* Cambridge University Press; 2021.

[30] Ghrist RW. *Elementary Applied Topology. Vol. 1*. Seattle (WA): Createspace; 2014. ISBN: 978-1502880857.

[31] Itskov V, Kunin A, Rosen Z. "Hyperplane neural codes and the polar complex." In: Baas N, Carlsson G, Quick G, Szymik M, Thaule M, editors. *Topological Data Analysis. Abel Symposia, vol 15.* Springer, Cham; 2018. pp. 343–369. Available at: https://doi.org/10.1007/978-3-030-43408-3_13.

[32] Potvin C. "Hypergraphs and their homology." PhD dissertation, Michigan State University; 2023.

[33] Myers A, Bittner A, Aksoy SG, Best DM, Henselman-Petrusek G, Jenne H, Joslyn C, Kay B, Seppala G, Young SJ, Purvine E. "Malicious cyber activity detection using zigzag persistence." *Proceedings of IEEE Conference on Dependable and Secure Computing Workshop on AI/ML for Cybersecurity,* 2023. Available at: https://ieeexplore.ieee.org/abstract/document/10354204.

[34] Gasparovic E, Gommel M, Purvine E, Sazdanovic R, Wang B, Wang Y, Ziegelmeier L. "Homology of graphs and hypergraphs." In: *Topological Data Analysis–Theory and Applications Workshop;* 2021 May 2. Available at: https://www.youtube.com/watch?v=XeNBysFcwOw.

[35] Kay B, Aksoy SG, Baird M, Best DM, Jenne H, Joslyn C, Potvin C, Henselman-Petrusek G, Seppala G, Young SJ, Purvine E. "Hypergraph topological features for autoencoder-based intrusion detection for cybersecurity data." In: *ICML Workshop on Machine Learning for Cybersecurity;* 2022. Available at: https://doi.org/10.48550/arXiv.2312.00023.

[36] Open Applied Topology. An open-source software package, not yet released.

Cybersecurity Anomaly Detection Using Graph Vertex Degree Assortativity

Daniel Juda

Two fundamental questions in cybersecurity are what is malicious activity and how do we detect it. These questions are generally answered using one of two paradigms. The first is signature-based threat detection, where known vulnerabilities are scanned using some form of detection software. The second is broadly termed anomaly detection, where analysts simply look for anything on the network that is considered atypical. We can distill this process down to asking the following: given network traffic during a fixed time period, can we determine if the character of the traffic is anomalous relative to some historical norm? Due to the constant change inherent in modern networks, establishing a historical norm can be difficult. Moreover, this constant change leads to many techniques highlighting behavior that is anomalous, but not actually malicious. There is a good deal of literature on these topics, too much to adequately summarize here, so we reference only the papers we borrowed techniques from. Our goal is to analyze internal netflow data from an enterprise network and identify subtle changes which may indicate malicious activity.

[Photo credit: iStock.com/carloscastilla]

Intuitively, network flow traffic lends itself to being encoded as a graph. This is formalized by Iliofotou, et al. in [1], where the authors describe a method for analyzing network traffic by considering directed graphs which they call traffic dispersion graphs (TDGs). These are graphs which consist of taking all nodes and edges in a network and then removing edges using some filtration criteria. We leverage this technique to study protocols independently. In particular, we look at well-understood protocols and analyze the TDG associated to each protocol using a well-known graph theoretic connectivity measure.

There are numerous techniques in graph theory to measure connectivity and communities. We utilize degree assortativity which was first introduced by Newman in [2, 3] and has since been used in a variety of fields. We use vertex assortativity on TDGs to identify days where subnetworks of an enterprise network are behaving atypically. We then use classical techniques from time series analysis to further investigate this behavior. The resulting combination of techniques provides a new method for analyzing network data and tipping of anomalies to cybersecurity threat analysts.

In this article, we first provide the necessary graph theory background for our technique, including an in-depth discussion of vertex assortativity. Next we discuss the necessary network theory background for our testing. Subsequent to that, we describe our testing and results, and we outline some ideas for using time series to drill down on detected anomalies. In the final section we offer conclusions. Suggestions for future directions are interspersed throughout the paper.

Graph theory background

A graph $G = (V, E)$ is a pair of sets, whose elements are called vertices and edges respectively, such that

$e \in E$ is a two element subset of V. If G is a directed graph, then edges are ordered pairs rather than unordered subsets of V. For each vertex $v \in V$ we call the values

$$d^{-}(v) = |\{e \in E|\ e = (v, x), x \in V\}| \text{ and}$$
$$d^{+}(v) = |\{e \in E|\ e = (x, v) \in E, x \in V\}|$$

the out-degree and in-degree of v respectively. The out-degree counts the number of edges originating at v and the in-degree counts the number of edges terminating at v. For each vertex $v \in V$, the total degree of v is $d(v) = d^{-}(v) + d^{+}(v)$, the count of edges either originating or terminating at v. When G is an undirected graph we use $d(v)$ to denote the degree of v. For general background on graphs see, for example, [4] for undirected graphs or [5] for directed graphs.

For a graph $G = (V, E)$, assortativity or the assortativity coefficient is a measure of the preference for the vertices of G to connect to other vertices that are similar under a chosen measure. The assortativity coefficient for G is a value r $\in [-1,1]$. If G has $r = -1$, then we say G is completely disassortative and if $r = 1$, then we say G is assortative. It was first defined by Newman in [2, 3] and has since been applied in a wide variety of fields such as biology [6] and social networking [7]. An extensive survey of assortativity results can be found in [8]. Throughout the paper, we will use the various degrees introduced above for our measure of similarity.

Let G be an undirected graph and p_k be the probability that a randomly sampled vertex from G has degree , that is, $p_k = \frac{1}{|V|}\ |\{v \in V \mid d(v) = k\}|$. Consider the distribution $q_k \propto p_{k+1}$ given by $q_k = \frac{(k+1)p_{k+1}}{\sum_j j p_j}$. This weighted distribution associates to k the number of times a vertex of degree $k + 1$ appears as an endpoint of an edge and is referred to as the remaining degree distribution. Intuitively, for an endpoint, v, of an edge, e, the remaining degree counts the number of edges incident to v other than e and is given by $d(v) - 1$. Let $\bar{d}$ nd s be the sample mean and sample standard deviation of the remaining degrees of the vertices. Let $f: E \to \mathbb{R}$ be defined as

$$f(e) = \frac{((d(v_i)-1)-\bar{d})((d(v_j)-1)-\bar{d})}{s^2}$$

where $e = (v_i, v_j)$. The assortativity coefficient of G is defined as

$$r = r(G) = \frac{1}{|E|}\sum_{e \in E} f(e).$$

The reader who is familiar with statistics will recognize this as a special case of the Pearson correlation coefficient of the degrees of the endpoints of a randomly sampled edge. If $s = 0$, then we define $r = 1$. This matches our intuition based on the definition, since if the standard deviation of the degrees is zero, then all edges connect to vertices that have the same degree.

The local assortativity at v, defined in [6, 9, 10] and denoted $\rho(v)$, is the amount v contributes

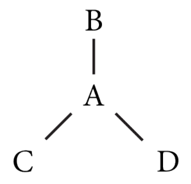


FIGURE 1. Example of a disassortative undirected graph. We have $d(A) = 3$ and $d(B) = d(C) = d(D) = 1$. This gives $q0 = q2 = \frac{1}{2}$, $\bar{d} = \frac{0+2}{2} = 1$, $s^2 = \frac{(0-1)^2 + (2-1)^2}{1} = 1$, and therefore $r = 1$. We also have $\rho(B) = \rho(C) = \rho(D) = -\frac{1}{6}$.

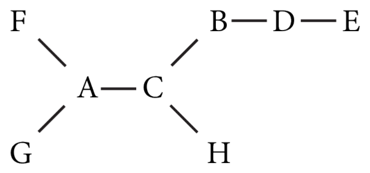


FIGURE 2. Example of an undirected graph. We have $d(A) = d(C) = 3$, $d(B) = d(D) = 2$, and $d(E) = d(F) = d(G) = 1$. This gives $q0 = q1 = q2 = \frac{1}{3}$, $\bar{d} = 1$, $s2 = 1$, and therefore $r = -\frac{2}{7}$.

to the global assortativity. Given this definition, $r = \sum_{v \in V} \rho(v)$. For $v \in V$, we denote by $N(V)$, the set containing v and all vertices adjacent to v called the neighborhood of v. To compute the local assortativity at v, we consider $H = (V_H = N(v), E_H) = G|_{N(v)}$, the induced subgraph on G restricted to the neighborhood of v. Using the notation above, we have

$$\rho(v) = r|_{N(v)} = \frac{\frac{1}{|E|}\sum_{e \in E_H} f(e)}{2}.$$

If , that is, the variance of the scaled vertex degree distribution is , then we define $\rho(v) = \frac{d(v)}{2|E|}$.

Let G be a directed graph. In [7], the authors extend the notion of assortativity to G by defining four different assortativity coefficients associated to G. Recall, for a vertex v in G, we have three notions of degree: in-degree, out-degree, and total degree. Also recall, that the purpose of assortativity is to measure the preference of vertices to connect to vertices that are similar. For undirected graphs, we used degree (or total degree) to measure similarity. In the case of a directed graph, we can choose which type of degree we want to use to measure similarity.

Let $Vi \subseteq V$ be the set of initial vertices and $Vt \subseteq V$ be the set of terminal vertices. Let $p^i_{k,+}$ be the probability that a randomly sampled vertex from V^i has in-degree k, that is, $p^i_{k,+} = \frac{1}{|V^i|}\,|\{v \in V^i \mid d+(v) = k\}|$. Notice, if v is the initial vertex of an edge e, then e contributes to the out-degree of v, not the in-degree. Thus we let $\bar{d}i$ and si be the sample mean and sample standard deviation of the in-degree. Thus we let $\bar{d}^i_+$ and s^i_+ be the sample mean and sample standard deviation of the in-degree of the initial vertices respectively. On the other hand, if v is the terminal vertex of an edge e, then e does contribute to the in-degree of v. Thus, for the terminal vertices we again consider the distribution $q^t_{k,+} \propto p^t_{k+1,+}$ given by $q^t_{k,+} = \frac{(k+1)p^t_{k+1,+}}{\sum_j jp^t_{j,+}}$. Let $\bar{d}^i_+$ and s^i_+ be the sample mean and sample standard deviation of the remaining in-degree of the terminal vertices respectively. Let f+,+: $E \to \mathbb{R}$ be defined as

$$f_{+,+}(e)\,\frac{\left(d^+(v_i) - \bar{d}^i_+\right)\left(\left(d^+(v_j)\right) - 1) - \bar{d}^t_+\right)}{s^i_+ s^t_+}$$

where $e = (v_i, v_j)$. Note, for the terminal vertices we account for the edge contributing to the in-degree and for the initial vertices, we do not. The in-in vertex degree assortativity coefficient of G is defined as

$$r_{+,+} = r_{+,+}(G) = \frac{1}{|E|}\sum_{e \in E} f_{+,+}(e).$$

The reader who is familiar with statistics will again recognize this as a special case of the Pearson correlation coefficient of the in-degrees of the endpoints of a randomly sampled edge. Similar definitions yield the in-out ($r_{+,-}$), out-in ($r_{-,+}$), and out-out ($r_{-,-}$) vertex degree assortativities. Suppose that $s^i_* = s^t_*$. If $\bar{d}^i_* = \bar{d}^t_*$, then we set $r_{*,*} = 1$. If the sample means don't agree, then we set $r_{*,*} = 1$. This again matches with our intuition based on the original definition. Intuitively, if there is no sample standard deviation and the sample means match, we have perfect correlation between initial and terminal vertices. On the other hand, if there is no sample standard deviation and the sample means don't match, we have perfect anticorrelation between initial and terminal vertices. Finally, if only one of s^i_* and s^t_* is 0, then we define $r_{*,*} = 0$.

Local assortativity was extended to directed graphs and discussed by Piraveenan, et al. in [6, 10]. It is once again defined, in each of the four different cases of directed assortativity, to be the contribution a vertex v makes to the global value. Thus, for example, the local in-in assortativity of a vertex v is given by $\rho_{+,+}(v) = r_{+,+}|_{N(v)} = \frac{\frac{1}{|E|}\sum_{e \in E_H} f_{+,+}(e)}{2}$, where $H = (V_H = N(v), E_H) = G|_{N(v)}$ as before. The other notions

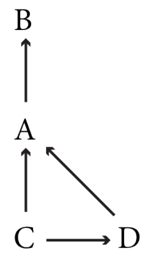


FIGURE 3. Example of a directed graph with in-in vertex degree assortativity $r_{+,+} = -\frac{\sqrt{2}}{8}$ We have *d^+ (C)=0*, $d^+(B)=d^+(D)=1$, $d^+(A)=2$. For the initial vertices, we have $V^i=\{A,C,D\}$, $p^i_{0,+} = p^i_{1,+} = p^i_{2,+} = \frac{1}{3}, \bar{d}^+_i = \frac{0+1+2}{3} = 1$ and $s^+_i = \frac{(0-1)^2+(1-1)^2+(2-1)^2}{2} = 1$ .For the terminal vertices, we have $V^t=\{A,B,D\}$, $q^t_{0,+} = q^t_{1,+} = \frac{1}{2}, \bar{d}^+_t = \frac{1}{2}$ and $s^+_t = \frac{1}{\sqrt{2}}$.

of directed local assortativity are defined similarly. Once again, if $s^i_* = s^t_* = 0$, then we define $\rho_{*,*}(v) = \frac{d^*(v)}{|E|}$ when $\bar{d}^i_* = \bar{d}^t_*$ and $\rho_{*,*}(v) = \frac{d^*(v)}{|E|}$ when $\bar{d}^i_* \neq \bar{d}^t_*$.

We close this section with some remarks on assortativity. It is generally observed that for large disassortative networks whose degree distribution follows a power law, such as an enterprise network or the Internet, the assortativity value decreases, that is, tends to 0. This is true in both the undirected and directed case. An alternative based on Spearman's Rho was proposed in [11, 12], but it is significantly more computationally intensive. We chose to limit the size of our graphs so that asymptotic behavior was not a concern and to avoid the additional computational complexity that is caused by using Spearman's Rho.

Cybersecurity background

Our goal is to apply graph theory to network flow (netflow) data to enable anomaly detection. Netflow data consists of metadata associated to transmission control protocol/Internet protocol (TCP/IP) connections. At the most basic level, a TCP/IP connection is a series of packets exchanged by the participants. In netflow data analysis, packets associated to a single communication are often combined into sessions. In particular, for our application a TCP/IP session between two hosts is a series of communications, such as a handshake followed by the transfer of desired information.

In order to use the graph theoretic techniques developed in the previous section, we need a well-defined method for transferring our netflow data to a graph. A traffic dispersion graph (TDG) as described in [1], is a graph, built from netflow data, with edges limited to those meeting some desired criteria. Formally, let G represent a set of network traffic where each vertex corresponds to an address[a] and each edge is a connection from a source address to a destination address. In particular, the edges are directed and G is a directed graph. We define a boolean function $f{:}E \rightarrow \{0,1\}$ for determining if an edge satisfies the chosen criteria. Our TDG is the subgraph $H=(V,E(H)) \subseteq G =$ such that $E(H)=\{\, e \in E \mid f(e)=1\}$. Thus, f functions as a filter for which edges are admitted to the subgraph H. In our application, we will filter the edges based on protocol using the standard TCP/IP port addressing scheme.

As an example, let's examine a protocol of interest: lightweight directory access protocol (LDAP). This protocol is a fundamental directory look-up protocol within a network used for things such as verifying a login name and password before allowing network access. In this example, f is defined piecewise as 1 if an edge represents a connection under the LDAP protocol and 0 otherwise. Thus $E(H)$ is the set of all edges that are a connection from a source IP address to a destination IP address under the LDAP protocol.

Within this protocol, a host (IP address) acts as either a server or a client. Typically, there is no peer-to-peer communication between clients and therefore all traffic on clients should be either to or from a server. Servers communicate to each other only when there is a need to utilize secondary servers due to traffic volume; that is, when a primary server receives too many access requests to handle, it will begin pushing excess access requests over to a secondary server. This will appear as "peer-to-peer" traffic between servers and may be an indicator of either a network malfunction or an attack such as a denial-of-service attack. Moreover, this will affect the assortativity value which we expect to be close to –1 in general for this TDG.

Application to the LDAP protocol

For our application, we consider netflow traffic from an enterprise network. Throughout this section, all graphs are directed as described in the graph theory background discussion. We made several choices to limit the amount of data being used to avoid the issues previously discussed. We limited our graphs

a. Throughout we will be using network layer addresses, that is, IP addresses. It is also reasonable to use physical layer Media Access Control (MAC) addresses.

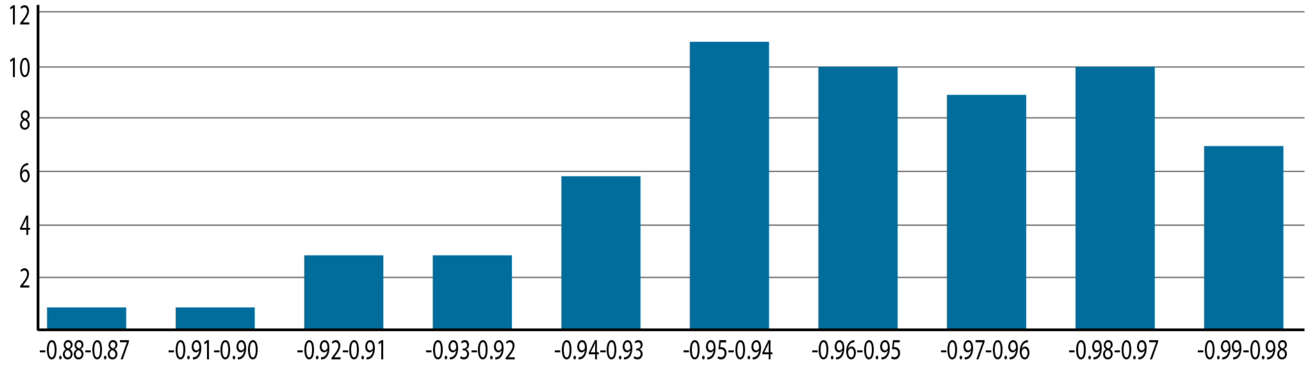


FIGURE 4. Example histogram of assortativity values built from 60 days of netflow using the LDAP protocol. Note the potential outliers in the [–0.87,–0.88] bin.

to subnets with the first three IP octets in common (commonly referred to as /24 subnets). We generate an edge between two hosts when there is a session between them, that is, we will not add edges for each packet in the session. We also only consider unique address pairings; that is, if a host initiates communication multiple times to the same destination, we only allow one edge between them in each direction. As an example, consider two IP addresses, *A* and *B*. We add two vertices *A* and *B* to our graph. If *A* initiates a session with *B*, then we add an edge (*A, B*) to our graph. If *A* later initiates another session with *B*, we do not add an edge. On the other hand, if *B* initiates a session with *A*, we add an edge (*B, A*).

Thus our workflow is the following. Fix a time-window from which to examine traffic and build TDGs, filtering based on LDAP, to represent each of several days. For each TDG, compute the directed assortativity coefficients. Given the four sets of coefficients, build a histogram to examine the distribution of each coefficient. We then visually isolate dates where one or more of the coefficients is anomalous. Figure 3 shows an example of a histogram of the in-out assortativity coefficients for a 60-day window on an enterprise network. Note, the values are clustered near $r= -1$ as expected. Two things are immediately clear from this histogram. Although our sample is small, if we considered the in-out assortativity value on a given day to be a random variable, this histogram suggests that the probability density function is far from any standard distribution. This makes hypothesis testing to determine if a value is anomalous more challenging since we do not have distributional information available. Despite this, the histogram suggests that there are some reasonable choices for outliers, such as the assortativity coefficients which fall into the [–0.87,–0.88] bin in the histogram. These outliers correspond to particular days.

Suppose we have identified a potentially anomalous day. We now want to establish a cause for this behavior. To do so we consider the local assortativity coefficients for each IP address seen in our graph. In figure 3 our anomalous day(s) appears to have an unusually high in-out assortativity value. Recall, the local assortativity coefficient for a particular vertex is that vertex's contribution to the global assortativity coefficient. Thus we are interested in individual vertices, or correspondingly IP addresses, that exhibit an unusually high local assortativity coefficient. This generates a set of IP addresses of interest for further study. These addresses can then be passed to a cybersecurity expert for examination.

A time series approach to analyzing and resolving anomalies

A time series is a collection of real-valued observations, $X=\{x_0, \dots , x_N\}$ made sequentially in time. We will briefly introduce some ideas in the area of time series analysis. For greater detail see, for example, [13]. Our goal is to use time series analysis to explain our anomalous IP addresses identified in the previous section on application to the LDAP protocol. To this end, we examine the time series of local assortativity coefficients for a particular IP address. We develop a time series model that best fits the series. We then use this model to forecast or predict the expected local assortativity values for the IP address and measure how far the predictions are from the corresponding true values.

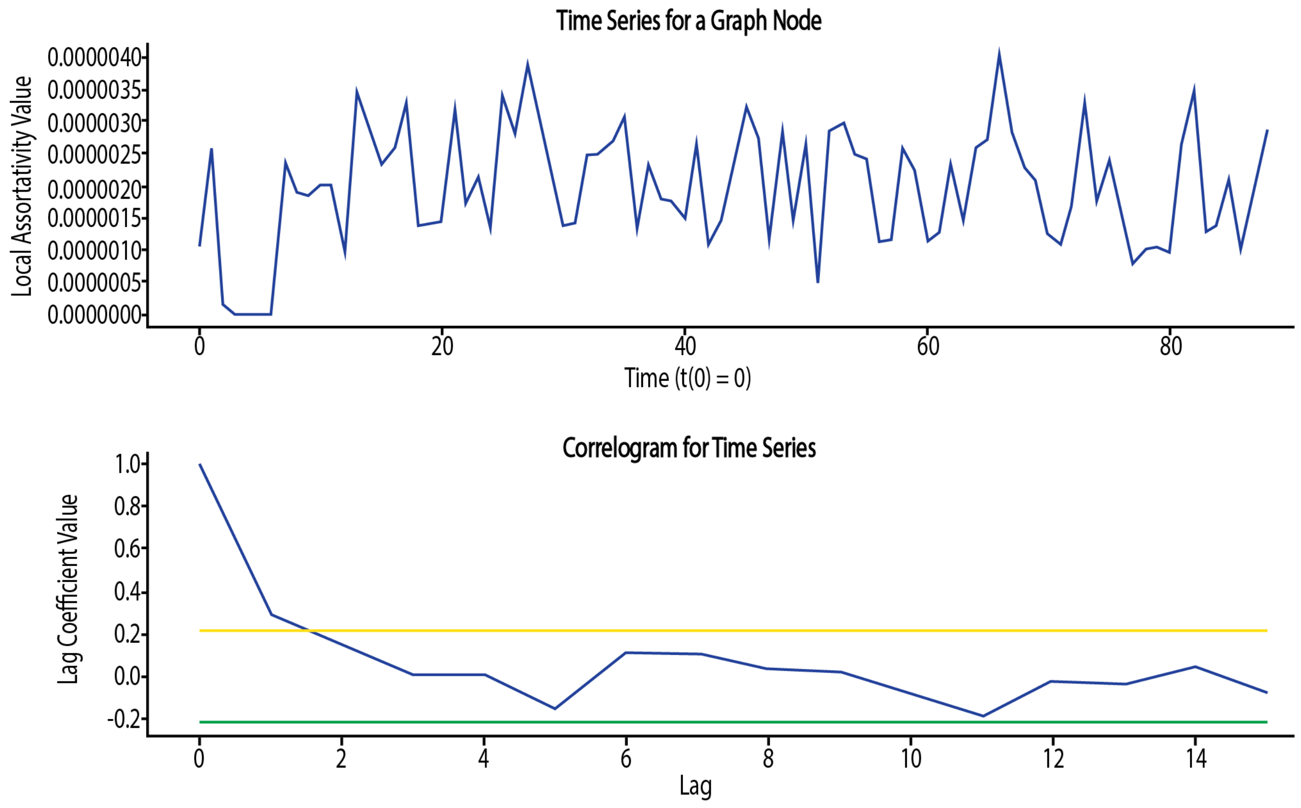


FIGURE 5. Example time series and correlogram of local assortativity values.

Given a time series $X=\{x_0, \dots, x_N\}$, we want to develop a model which can be used to forecast or predict series values. For X, the lag correlation coefficient of lag k measures how well correlated values that are k units of time apart in the time series are. It is given by

$$\boldsymbol{r_k} = \frac{\sum_{i=0}^{N-k}(\boldsymbol{x_i}-\bar{\boldsymbol{x}})(\boldsymbol{x_{i+k}}-\bar{\boldsymbol{x}})}{\sum_{i=0}^{N}(\boldsymbol{x_i}-\bar{\boldsymbol{x}})^2}$$

Where $\bar{x} = N^{-1}\sum x_i$ is the average value of the time series. These values can be plotted in a correlogram and are useful for determining a model type that is likely to fit the data. Figure 5 shows a plot of a time series of the local assortativity coefficients of an individual IP address for 100 days and the corresponding correlogram. Recall, our goal is to use the time series of local assortativity coefficients to explain anomalous IP addresses. To do so, we identify a best-fit time series model of our data and then consider the residual error as a test statistic for quantifying how anomalous the behavior of the IP address is.

Under suitable assumptions, for a random time series, the lag correlation coefficients are approximately normally distributed with mean 0 and variance $\frac{1}{N}$. Thus, in the random case we expect the majority of the lag correlation coefficients to fall in the interval $\left[-\frac{2}{\sqrt{N}}, \frac{2}{\sqrt{N}}\right]$. The horizontal lines in the correlogram of figure 5 are placed to mark this interval. Notice that the autocorrelation coefficients initially decrease monotonically to $r_3=0$ and $r_i \approx 0$ for $i > 3$. This suggests that an autogressive (AR) model would be a reasonable choice of model. We choose an AR model of order three given $r_3=0$, that is, the series value at time t, depends linearly on the values at times $t-1$, $t-2$, $t-3$. Thus, a framework for our model is given by

$$x_{t+3} = \alpha_0 x_t + \alpha_1 x_{t+1} + \alpha_2 x_{t+2} + \epsilon,$$

where $\epsilon \in \mathbb{R}$ represents some small error value that is normally distributed with mean 0. The α_i are parameters that are estimated by the method of least squares to find the model which gives the best approximation of the original time series.

Once we have found the coefficients for our model, we can generate the new series of predicted values $\{\widehat{x_3}, \dots, \widehat{x_N}\}$with the value $\widehat{x_\iota}$, corresponding to in the original series. Moreover, we can generate a series of residual error values $\mathcal{E}=\{e_i= \widehat{x_\iota} - x_i \mid i=3, \dots, N\}$, which we can think of as a series of random variables, and consider the empirical probability density function for $\mathcal{E}$

Under idealized circumstances, that is, we know the true values for the and the choice of model is perfect, we would have $e_i = \epsilon$ and therefore $\mathcal{E} \approx N(0,\sigma_\epsilon)$. Unfortunately, in practice, the coefficients α_i are approximations. Thus we have to be careful in how we choose to use the e_i for hypothesis testing in general. Fortunately, if the model is chosen well, then in most applications $\mathcal{E}$ tends to be approximately normal. This fact can be leveraged to apply standard hypothesis testing techniques. For example, we can use Z-score testing on the empirical distribution of the residual errors to find outliers which correspond to days when the local assortativity coefficient of the IP address is anomalous.

Conclusion

We developed an application of graph theoretic and time series analysis techniques to answer questions about anomalies in the cybersecurity domain. This work can be used to develop a semi-automated workflow for monitoring an enterprise network. We introduced some interesting directions for future work including using alternative connectivity measures.

References

[1] Iliofotou M, Pappu P, Faloutsos M, Mitzenmacher M, Singh S, Varghese G. "Network monitoring using traffic dispersion graphs (TDGs)." In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC '07);* 2007; New York (NY): Association for Computing Machinery: pp. 315–320. Available at: https://doi.org/10.1145/1298306.1298349.

[2] Newman MEJ. "Assortative mixing in networks." *Physical Review Letters.* 2002;89(20):208701. Available at: https://doi.org/10.1103/PhysRevLett.89.208701.

[3] Newman MEJ. "Mixing patterns in networks." *Physical Review E.* 2003;67(2):26126. Available at: https://doi.org/10.1103/PhysRevE.67.026126.

[4] Diestel R. *Graph Theory.* Springer-Verlag, 2005. ISBN: 9783540261834.

[5] Bang-Jensen J, Gutin G. *Digraphs: Theory, Algorithms, and Applications.* Springer-Verlag, 2001. ISBN: 9781852332686.

[6] Piraveenan M, Prokopenko M, Zomaya A. "Assortative mixing in directed biological networks." *IEEE/ACM Transactions on Computational Biology and Bioinformatics.* 2012;9(1):66–78. Available at: https://doi.org/10.1109/TCBB.2010.80.

[7] Foster JG, Foster DV, Grassberger P, Paczuski M. "Edge direction and the structure of networks." In: *Proceedings of the National Academy of Sciences,* 2010 Jun 15;107(24):10815-20. doi: 10.1073/pnas.0912671107.

[8] Noldus R, Van Mieghem P. "Assortativity in complex networks." *Journal of Complex Networks.* 2015;3(4):507–542. Available at: https://doi.org/10.1093/comnet/cnv005.

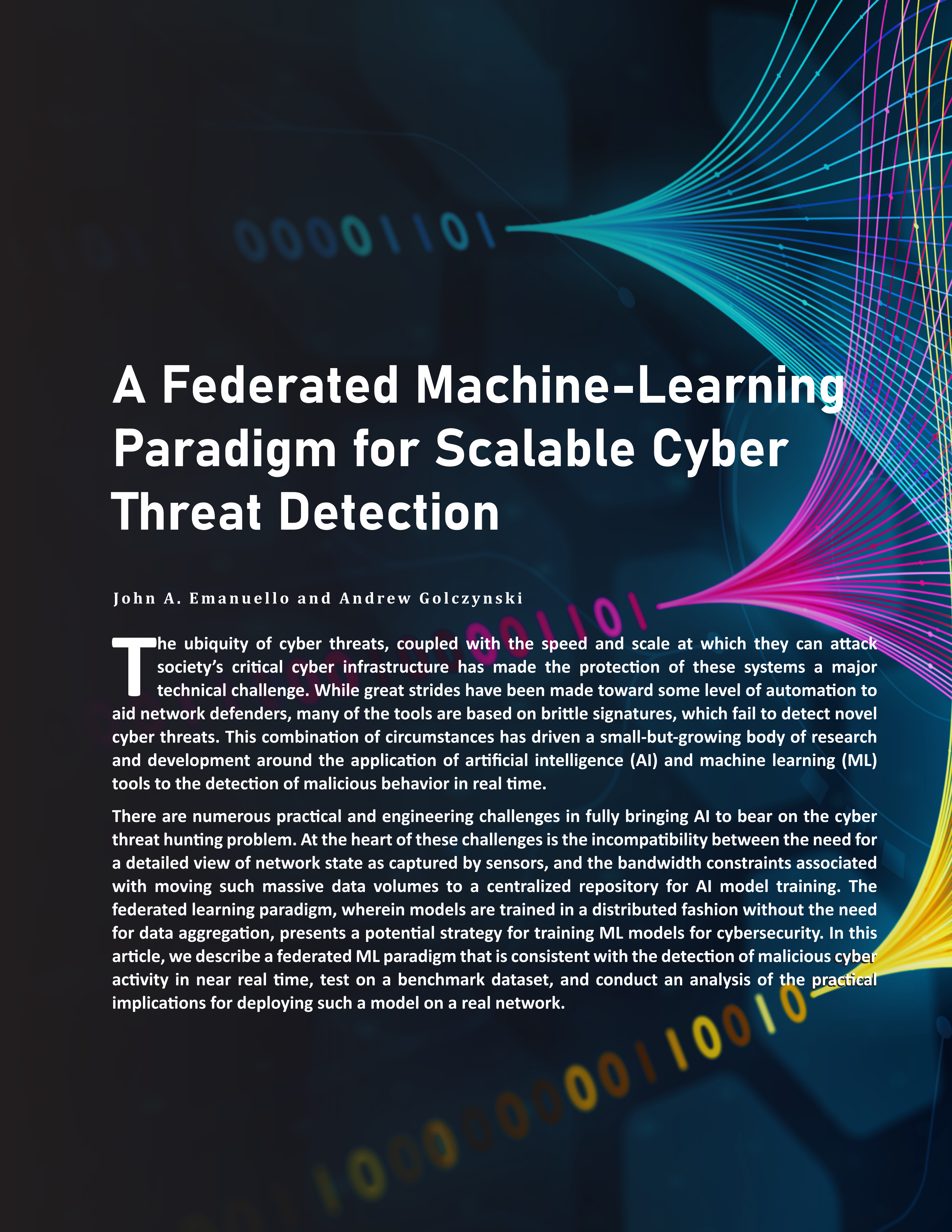
[9] Piraveenan M, Prokopenko M, Zomaya AY. "Classifying complex networks using unbiased local assortativity." In: *Artificial Life XII: Proceedings of the 12th International Conference on the Synthesis and Simulation of Living Systems, ALIFE;* 2010. pp. 329–336.

[10] M. Piraveenan, Prokopenko M, Zomaya AY. "Local assortativeness in scale-free networks." *Euro-Physics Letters.* 2008;84(2). doi: 10.1209/0295-5075/84/28002.

[11] Hoorn P, Litvak N. "Degree-degree dependencies in directed networks with heavy-tailed distributions." *Internet Mathematics.* 2015;11(2):155–179. Available at: https://doi.org/10.1080/15427951.2014.927038.

[12] Litvak N, Hofstad R. "Uncovering disassortativity in large scale-free networks." *Physical Review E.* 2013;87(2):22801–22808. Available at: https://doi.org/10.1103/PhysRevE.87.022801.

[13] Chatfield C. *The Analysis of Time Series An Introduction.* Chapman and Hall/CRC: 2004.



A Federated Machine-Learning Paradigm for Scalable Cyber Threat Detection

John A. Emanuello and Andrew Golczynski

The ubiquity of cyber threats, coupled with the speed and scale at which they can attack society's critical cyber infrastructure has made the protection of these systems a major technical challenge. While great strides have been made toward some level of automation to aid network defenders, many of the tools are based on brittle signatures, which fail to detect novel cyber threats. This combination of circumstances has driven a small-but-growing body of research and development around the application of artificial intelligence (AI) and machine learning (ML) tools to the detection of malicious behavior in real time.

There are numerous practical and engineering challenges in fully bringing AI to bear on the cyber threat hunting problem. At the heart of these challenges is the incompatibility between the need for a detailed view of network state as captured by sensors, and the bandwidth constraints associated with moving such massive data volumes to a centralized repository for AI model training. The federated learning paradigm, wherein models are trained in a distributed fashion without the need for data aggregation, presents a potential strategy for training ML models for cybersecurity. In this article, we describe a federated ML paradigm that is consistent with the detection of malicious cyber activity in near real time, test on a benchmark dataset, and conduct an analysis of the practical implications for deploying such a model on a real network.

[Photo credit: iStock.com/NicoElNino]

Background

The difficulty of cybersecurity lies in the asymmetric advantages the adversary holds. Indeed, defenders must protect all assets against all their possible vulnerabilities; whereas, an attacker need only exploit a single vulnerability to gain access on a network and set into motion a multistage cyberattack. Armed with a seemingly unlimited number of tools, advanced persistent threats (APTs) are highly sophisticated and motivated, enabling them to relentlessly target our most sensitive networks, including those that support critical infrastructure.

APT-style attacks are rarely composed of a single action; rather, they typically contain numerous events which constitute multiple stages of an attack. The progression of these stages is often modeled by ontologies such as Lockheed Martin's Cyber Kill Chain, which tracks and organizes detectable behaviors from individual log events or collections thereof [1, 2]. The subtlety of attacker behaviors, especially from sophisticated actors, makes detection difficult, with discovery often only happening after the goals of the attack have been accomplished.

It is worth mentioning that this is not an abstract problem. Digital systems underpin nearly all aspects of modern societies, including economic institutions, critical infrastructure, and even democracy itself. From a commercial standpoint, malicious cyber actors are motivated by economic espionage, especially the theft of intellectual property, which by some estimates can cost a company billions of dollars in revenue losses alone [3]. For democracies, cyberattacks could disrupt elections and erode public confidence in democratic institutions. As such, robust measures to secure cyber systems are critical components of safeguarding societal stability, economic resilience, and national security.

Intrusion detection systems

The evidence of attacks on an enterprise network can be captured by a high number of disparate sensors, logging network traffic, cloud telemetry, end point activity, etc. As such, development of automated approaches to intrusion detection have been an area of interest at least since the late 1980's [20]. These intrusion detection systems (IDS) have historically relied heavily on signature-based rules, which describe a known malicious activity that, upon matching an observed behavior, alert a human defender to investigate. These rules include byte patterns in network traffic packets or attribute patterns in host logs [4, 5].

However, as these rules can only describe known malicious behavior, APTs can easily defeat these signatures. For example, if a signature includes a specific rule involving a byte pattern in network packets, the adversary can simply fragment packets or otherwise change the contents of packets to defeat the rule and still perpetrate the attack. As such, these signatures must be constantly updated in an at least partially manual process that often consumes more time than the adversary needs to implement a countermeasure. Further in the adversary's favor is the wide availability of such rules to exploit in order to misdirect defenders or otherwise obfuscate their attacks; indeed, by overloading defenders with alerts, they can make it extremely difficult for defenders to decide what to focus on. All told, these drawbacks are driving a body of work to apply ML to create more flexible iterations of IDS.

Given the extremely high degree of variance of normal user activity, these models must be trained on data which sufficiently captures the network's baseline, and this can only be done when data is sampled at a high rate across the assortment of sensors. Contrary to centralized ML paradigms, the data volumes and bandwidth constraints of an enterprise network prohibit the ingestion of sufficient amounts of data to a single compute server. These unique constraints suggest that federated learning may be a potential paradigm for training autonomous threat hunting tools. More specifically, such approaches support the training of ML models on multiple compute nodes, each with their own training data (potentially drawn from different distributions) that feed a global model that can be deployed at the edge for inference.

It should be noted that, given the evolving nature of cyber threats and the high degree of variance of normal activities between different networks, the availability of a labeled dataset would not be conducive to the task at hand. Thus, it is far more effective to train these ML models in unsupervised fashion, as anomaly detectors, rather than as explicit malicious behavior classifiers. Algorithms such as clustering, isolation forests, and autoencoders are well suited for anomaly detection and are integrated in widely used software packages, making them easy for researchers and cybersecurity analysts to use.

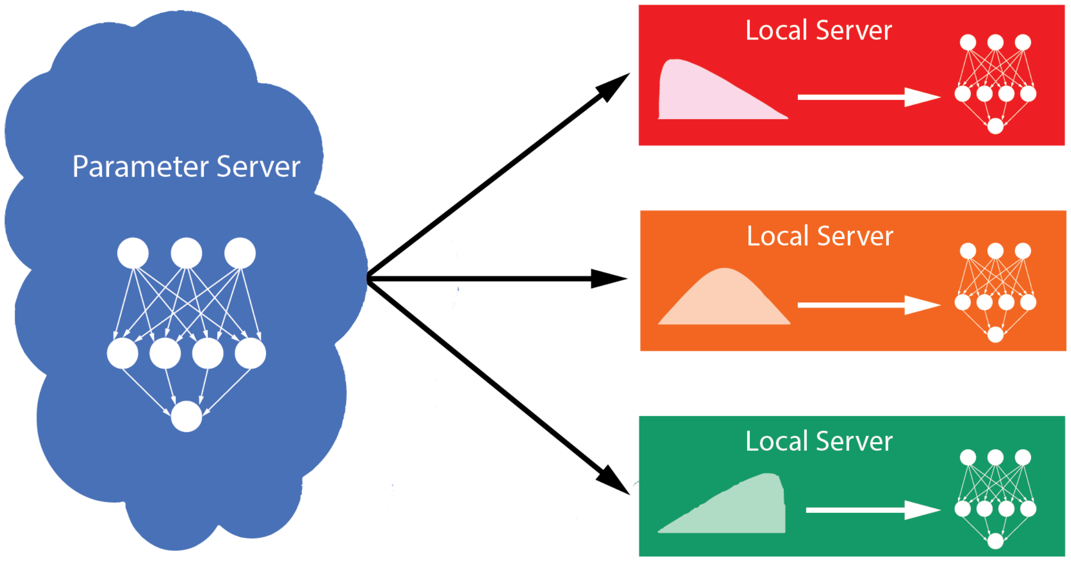


FIGURE 1. Federated learning is a machine learning setting where multiple entities (i.e., local servers) collaborate to train a model, under the management of a central parameter server. Each client's raw data is stored locally and is not exchanged or transferred; instead, local servers complete some number of training steps on the model, and their feedback is aggregated by the central server that serves an updated model for either further training by the local servers or deployment.

Neural network approaches for intrusion detection

In the approach we outline below, we appeal to an autoencoder (AE), which is a neural network that is designed to be an identity function on the input space, and is composed of two functions: an encoder and a decoder [6]. More specifically, the training process of an autoencoder $f = g \circ h$ results in a nonlinear encoder $h\colon \mathbb{R}^n \rightarrow \mathbb{R}^m$ and decoder $g\colon \mathbb{R}^m \rightarrow \mathbb{R}^n$ functions, where $m \ll n$ and the objective function is mean squared error loss with respect to the standard Euclidean distance between input and output: $\|x - f(x)\|_2$. We note that this choice of relative dimension sizes results in what is called an "undercomplete autoencoder" and is, in effect, a nonlinear analogue principal component analysis [6]. By construction, (undercomplete) AEs learn salient information about the training data, with the compression/reconstruction process failing on outliers or other data drawn from distributions vastly different from that of the training set. This property allows an AE to indicate when a given datum was drawn from a novel distribution, making the architecture ideal for anomaly detection [7, 8].

The choice of a neural network as a proposed architecture for detecting malicious cyber activity has been studied in the literature, and shows promise as a more flexible alternative approach from traditional signature-based detection schemes [9, 10]. While cyber telemetry logs largely contain categorical data points, there are techniques such as *log2vec* that can transform these logs into numerical data points that are ingestible by neural network architectures [9, 11, 12].

Our prior work demonstrates that the numeric representations of cyber logs can be learned in tandem with detection tasks [12]. Here, our NLP-inspired approach to anomaly detection requires us to embed our preprocessed records into a semantically relevant vector space, which is the input to an AE-based anomaly detector at inference. We tested this technique on the Operationally Transparent Cyber (OpTC) dataset created by Defense Advanced Research Projects Agency (DARPA) [17, 18], which consists of network and host logging collected from a network of hundreds of Windows hosts over a one-week period. The activity represents both normal (benign) user activity, as well as logging from an APT-inspired red team attack over the course of three days, during which numerous machines were under different phases of a multi-staged attack. Our technique was able to detect high volumes of anomalous activities during the red team attack, and very low volumes of anomalies when there were no attacks.

A federated-learning paradigm for intrusion detection

Federated learning (FL) is a relatively new concept in the field of AI/ML that involves training a model across multiple devices, which are often called nodes or learners. Each node trains the model separately on its own data and shares model information with a central server [13]. This central server then aggregates the contributions of the participating nodes to produce a model, which is shared back to the nodes for further training or inference. There are several flavors of training archetypes (e.g., synchronous and asynchronous) and aggregation techniques which are covered in the literature. For our purposes, we assume that the model in question is some kind of neural network and that a single training step of the model is as follows:

1. The central server transmits the most current model to each of the local servers.
2. Each local server feeds a portion of its local data through the model and computes the gradients of the mutually agreed upon objective function and transmits these gradients back to the central server.
3. The central server aggregates the gradients from the local servers and performs a model update.

Federated ML presents a viable solution by distributing the learning process to the network's edge, where the data is generated. This paradigm balances the need for models to be trained on realistic, live data with the practical constraints presented in cybersecurity [13, 14]. Rather than transmitting raw data to a centralized server, federated learners utilize local computation and collaboration among networked devices to train AI models. This decentralized approach significantly reduces the need for transferring large amounts of data across the network, alleviating bandwidth constraints and minimizing latency issues.

Another consideration is the false positive problem. Indeed, if AI/ML is to be a force multiplier in cybersecurity, any solution must be careful not to produce more false positives than a human analyst can adjudicate. In fact, a major problem with current systems is that the number of alerts defenders must sift through are unmanageable. If AI/ML solutions are to attain wide adoption, they must add analytic value rather than generate unhelpful alerts. In a FL paradigm, training on a wider variety of data facilitates better approximation of the true distribution of the "data in the wild." However, in a FL paradigm, the nonuniform occurrence of certain data points across the unified training set allows federated models to still learn salient information about these points, and to pass those insights on to a deployed model.

To see the importance of such a consideration in cybersecurity, note that normal activities vary widely across users; for example, certain users have a higher propensity to run programs like Microsoft Excel than others. This means that, for an ML model trained on host-based logs, if a user who rarely launches Excel simply does so, the model would mark the behavior as anomalous. However, simply launching Excel is hardly indicative of a cyberattack. Hence, in a FL paradigm, it is possible to aggregate the vast number of normal activities within a model, in a manner that is reminiscent of balancing the "bias-variance trade-off" [15].

Given these strengths, we propose that an FL paradigm could be employed in protection of an enterprise computer network to discover potentially malicious activity in a way that is both robust to evolving cyber threats and capable of alleviating alert fatigue that exists due to current rules-based IDS.

Early after its inception, a key selling point for federated models was their purported ability to keep local data private [13, 14]. As cyber systems often contain sensitive information such as personally identifiable information (PII), proprietary information/intellectual property, or any information that could cause harm if released in the public domain, one could imagine that FL techniques could be employed across organizations to produce community-driven AI/ML models for cybersecurity, without risking the disclosure of the sensitive information associated with those systems.

However, as noted in the literature, such privacy claims have been grossly overblown [15, 16]. Indeed, ML model updates can leak several varieties of information to even casually motivated attackers. More specifically, attackers can infer properties of the training data or even reconstruct specific training examples. While the amount of work required to extract this sort of information from ML models can be quite expensive, the economics of cybersecurity make it so that the information to be extracted is worth the resources. Absent advances in techniques such as homomorphic encryption, we believe that the technology is not sufficiently able to mitigate the risks associated with cross-organizational federated learners.

Our approach

We implement a FL paradigm across five different servers, each of which stand in for a single host on an enterprise network. Each server contains data from exactly one host in our dataset, and no data is shared across servers. Using Tensorflow's distributed MultiWorkerMirrored strategy, we instantiate a common model across five g4dn.8xlarge AWS EC2 instances, with one of the workers also playing the role of the global server for aggregating gradients and serving model updates.

The precise model architecture is an implementation that is similar to that of our previous work [12]. The input is a log record corresponding to single network or host activity on a computer (e.g., opening a network flow, editing a registry key, etc). Each record consists of 27 fields (e.g., user name, IP address, path, registry key, etc.), each of which can take on any of a variety of values, which we shall call "words." Each of the 27 words y_i of the record is fed into an embedding layer to obtain a vector representation of each in $\mathbb{R}^8$. The 27 eight-dimensional word embeddings are then concatenated into a 216-dimensional vector V, which corresponds to the embedding of the record. The record embedding is fed into an AE with a single hidden layer mapping the record into $\mathbb{R}^8$ and back to $\mathbb{R}^{216}$. This reconstruction of the record embedding, $\hat{V}$ is the first output of the model. We then break $\hat{V}$ back into its 27 component vectors, each of which is fed into a common dense layer to effectively compute word-by-word probability distributions across the entire vocabulary space $(\hat{y}_i)$. The objective for the neural network is a two-term loss function, which balances the ability of the network to reconstruct the record embeddings with its ability to predict the component words that constructed the record from the reconstructed embedding. More succinctly, the record-level loss is $Loss = \sum_{i=1}^{27} CEL(\hat{y}_i, y_i) + \alpha \| V - \hat{V} \|$, where CEL is Tensorflow's sparse categorical cross entropy loss from logits and α is a regularization term to keep the reconstruction loss of the internal AE roughly on par with the CEL term. We experimented with different values and settled on $\alpha = 100$ for this task, which puts the two loss terms on roughly equal magnitude. Note that this is different from our previous work, where we used $\alpha = 5$ for data drawn from single hosts.

Results from our previous work demonstrate that the model is both effective at learning semantically relevant representations for the words in the records as well as providing a useful anomaly detection scheme on cybersecurity logs. Our goal here is to demonstrate how such a model could be employed in federated fashion to learn on an even larger scale than previously demonstrated, and to demonstrate that such models remain effective at the downstream task of detecting malicious cyber activity in the OpTC dataset.

OpTC dataset

As in our previous work, we appeal to the OpTC dataset. As mentioned above, these data consist of network and host logging collected from a network of hundreds of Windows hosts over a one-week period, representing normal (i.e., benign) user activity [17, 18]. Over a three-day period within this week, a set of red team actors also worked to perform various penetration tests against the network, seeking to infiltrate the network, ensure persistence, and carry out increasingly complex attacks over time. As the data contains logs from hosts that were and were not attacked by the red team, and each host that was attacked was only attacked during a proper subperiod of the three-day attack, this data constitutes a strong benchmark for intrusion detection techniques.

Specifically, we used the ECAR collection (so named as its format is an extension of the Mitre Cyber Analytics Repository, or CAR, data format), which consisted of combined network and host metadata logging drawn from Sysmon, Procmon, and other sensors [19]. While each record contained only a handful of fields, 58 unique fields were present across the entirety of the raw data, including the network five tuple (source and destination IPs/ports and protocol), image and module paths, and usernames. To prepare the data for use in our experiments, we applied a very-light handed preprocessing approach to the logs from five hosts, namely sysclient 0201-0205 to create a vocabulary consisting of terms that were tokenized. We chose to follow the exact same preprocessing pipeline, except where noted below.

These steps resulted in a vocabulary size of 10,667 words across the training and test corpora:

1. **Dropping unnecessary features:** Of the 58 keys in the ECAR file, only 27 were kept for processing. The remaining fields were determined to be insufficiently relevant to our chosen task/approach and were dropped.
2. **Feature prefixing:** Some terms may have specific meanings depending on which feature they are associated with. For example, the number 443 has a specific meaning in the “dest port” field, but that meaning would not be preserved in other fields. To ensure that these meanings are respected, values of select features were prefixed with their feature name (e.g., “443” would be mapped to the term ”PORT_443” when it is found in the “dest port” field).
3. **Connection time bucketing:** Network activity records contain start and end times for the connection; to discretize these values, connection durations were calculated and bucketed into SMALL, MEDIUM, and LARGE buckets based on manual inspection of the distribution of connection durations.
4. **Path/file name extraction:** File paths and registry keys can contain machine-specific sets of directories, even when considering a common item (e.g., “xyz.dll” might be found in different directory locations on separate machines, even though the underlying file is the same). To correct for this, such paths are reduced to only the file name. Similarly, when parsing command line input, only the name of the executed program is maintained (i.e., the full path and any arguments are dropped).
5. **/24 CIDR subnet extraction:** IPv4 addresses were, in many cases, too sparse across the dataset to be well-represented, so we opted to remove the last octet from each address and only use the /24 subnet. While this did reduce the resolution of the representation of our addresses, we gained additional robustness by aggregating rare terms into a smaller number of representations.
6. **Ephemeral port aggregation:** Ports used for outgoing network traffic are typically arbitrarily chosen from the high end (i.e., greater than 49151) of the range of valid ports. These choices carry no real meaning, and needlessly expand the vocabulary space, and so such ports were replaced with a generic ”EPHEMERAL_PORT” token.
7. **Removal of rare terms:** After performing the above tokenization, terms that appeared fewer than 10 times (determined by examination of the distribution of term frequencies) over our training corpus were discarded and replaced with a generic ”OBSCURE_TERM” token.
8. **Conversion to vocabulary indices:** All remaining vocabulary tokens were indexed, and each token was replaced with its index. Empty fields (e.g., file path fields are empty for network traffic events) were replaced with a ”NULL_TERM” token.

With the vocabulary determined, we needed to select a training corpus for the neural network model described above. Our goals were primarily to measure the efficacy of the model in detecting red team activity, and to assess the model’s ability to generalize to normal activities rarely-or-never seen by the model. To accomplish this, we chose to train on sysclient 0203, sysclient 0204, and sysclient 0205. The choice of these three hosts is primarily motivated by the fact they had the largest volumes across the five hosts, at roughly 24 million records each (the other two hosts had roughly 10 million fewer records available). Given the data volumes, we restricted the training set to only include the three-day period preceding the red team events. Test data constituted the three-day period corresponding to the red team exercise, during which sysclient 0201, sysclient 0203, and sysclient 0205 were attacked. Sysclient 0202 and sysclient 0204 contain no red team activity, and we include model inference results from these hosts for comparison. In sum, our test set included roughly 45 million records across the five hosts.

We also note that, due to restrictions in Tensorflow’s MultiWorkerMirrored distributed learning strategy, we had to ensure that every node had precisely the same batch size and same number of batches. We chose to limit the number of records from each node in the training set to be precisely 24 million records.

Neural network training paradigm

We do wish to note one key difference in the training paradigm between this and our previous works. Previously, we constructed a custom Tensorflow training loop within Python, wherein we selected

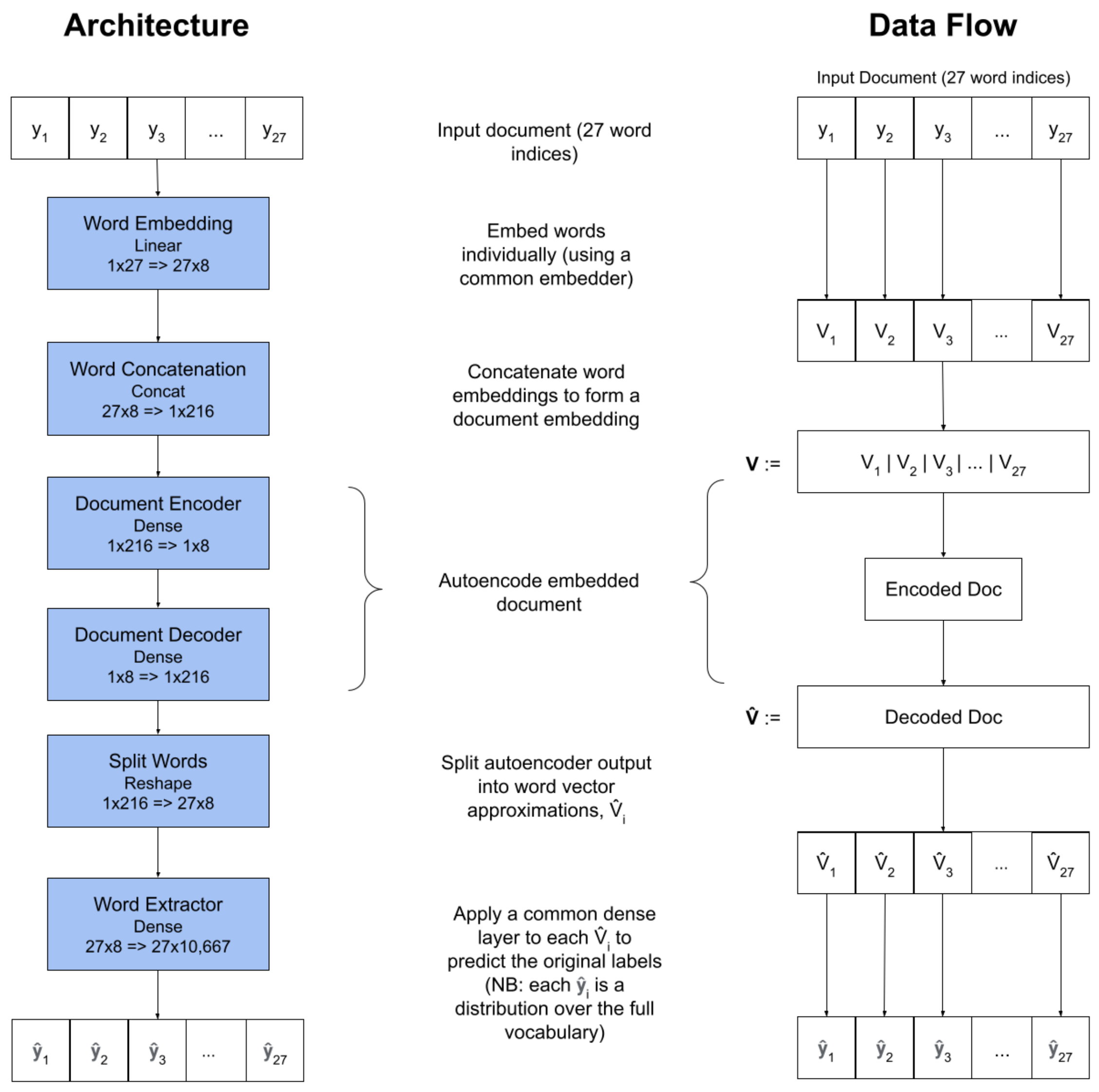

Architecture
Data Flow
Input Document (27 word indices)
y1 y2 y3 ... y27
Word Embedding Linear 1x27 => 27x8
Word Concatenation Concat 27x8 => 1x216
Document Encoder Dense 1x216 => 1x8
Document Decoder Dense 1x8 => 1x216
Split Words Reshape 1x216 => 27x8
Word Extractor Dense 27x8 => 27x10,667
ŷ1 ŷ2 ŷ3 ... ŷ27
Input document (27 word indices)
Embed words individually (using a common embedder)
Concatenate word embeddings to form a document embedding
Autoencode embedded document
Split autoencoder output into word vector approximations, V̂i
Apply a common dense layer to each V̂i to predict the original labels (NB: each ŷi is a distribution over the full vocabulary)
V1 V2 V3 ... V27
V := V1 | V2 | V3 | ... | V27
Encoded Doc
V̂ := Decoded Doc
V̂1 V̂2 V̂3 ... V̂27

FIGURE 2. Each compute node conducts a training step on a replica of this model, which is described in the text. The total number of trainable parameters is 185,189, of which 85,416 are for vocabulary embeddings. The remaining 100,000 parameters are spread across the encoder, decoder, and word extractor layers, the last of which constituted 96,093. Thus, the model puts a strong emphasis on producing effective vocabulary that are informed by the semantic relationships within the data.

batches based on the inverse weight of the occurrence of terms appearing in the corpus. The intention behind that choice was to ensure that the embeddings of rarely occurring terms were given sufficient opportunity to converge, while not overfitting the overall model to frequently occurring terms. This custom training loop did come at the expense of speed; however, when we attempted to implement the same approach using Tensorflow's distributed MultiWorkerMirrored strategy, the latency associated with model update sharing made the federated training process prohibitively slow. To circumvent this issue, we reworked the training process, and trained the FL models with model.fit() using batch sizes of 512 across each of the compute nodes. While we did notice a significant slowdown in training time (as compared training with model.fit() on a single worker), the overall speed was still reasonable.

On a related noted, our selection of EC2 hardware was the result of some early experiments, wherein

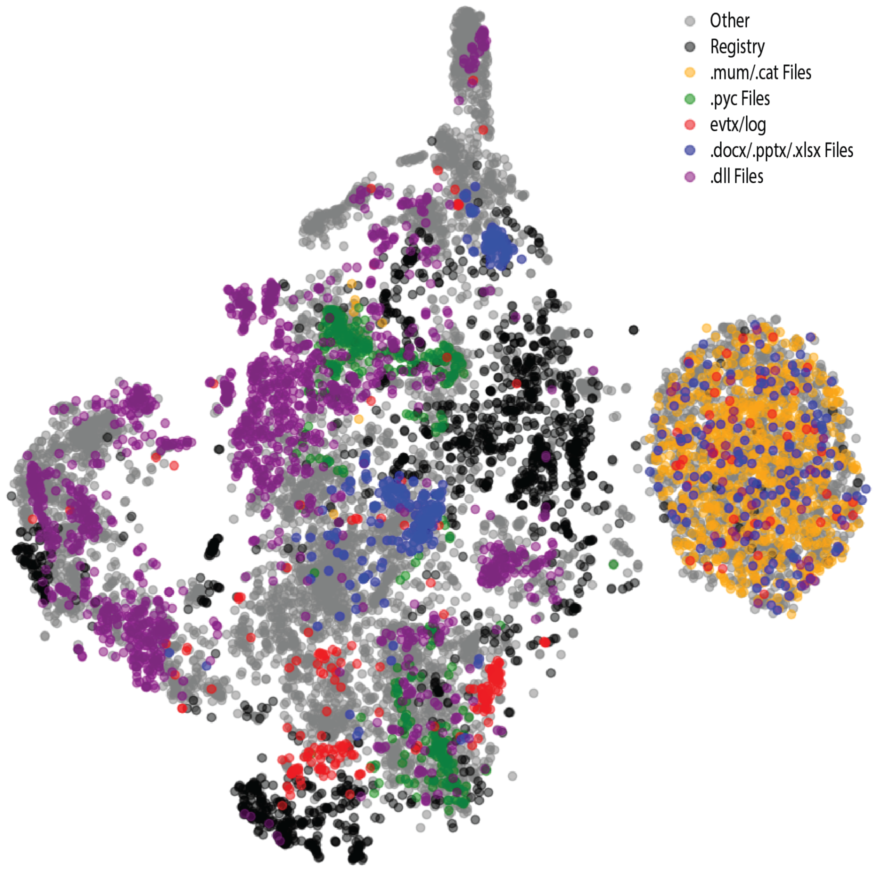


FIGURE 3. These word embeddings show tight clusters around many different types of file types that appear commonly across the dataset (e.g., Microsoft Office files in blue). Note these are a t-distributed stochastic neighbor embedding (TSNE) of the original eight-dimensional vectors down to two dimensions.

we tested this methodology on two g3.xlarge servers and found the training process to be too slow. We expect that this was due to the rather low network performance of these instances (i.e., up to 10 gigabits per second) and opted instead for the g4dn.8xlarge, which provided a guaranteed 50 gigabits per second. The training process here appeared to be much faster, but was still slower than that observed in the nondistributed case.

Results

Our testing methodology reflects our previous work, so that the FL model may be compared to the individual models trained on each of the hosts we examined. This methodology is consistent with using these types of utilities to alert administrators at times when network and host logs are exhibiting anomalous behavior, and to provide prioritized lists of anomalies for investigation/action in an operational environment. In cybersecurity use cases, overall accuracy is not a useful measure for our performance, as we have a large class imbalance (i.e., less than one percent of our test data is labeled as anomalous). Our goal is not necessarily to correctly classify all of our malicious traffic as anomalous, as some of that traffic may be only incidentally labeled as malicious due to its association with a red team process. Instead, we choose to provide two main results that address the above use cases: a qualitative view of changes in the network's cross-entropy loss over time, and measurements of precision using various error levels as classification thresholds. We feel that these results best promote our goal of allowing administrative users to know when an attack is taking place, while also providing a manageable number of true positives, with a high degree of certainty (i.e., high precision).

In terms of assessing the detection efficacy of the model, we compute an anomaly score s for each record, which as in our previous work is

$s = \sum_{i=1}^{27} CEL(\hat{y}_i, y_i)$, as a measure of how well the model reconstructs the input data. In the ideal, the model should be able to reconstruct benign data with higher fidelity than data associated to red team events.

Overall, the results of these measures indicate that the model is capable of distinguishing benign activities from red team events. Additionally, figure 3 shows the learned embeddings for the vocabulary do result in semantically consistent embeddings. That is, key categories of vocabulary words (e.g., registry keys/values, mum/cat files) and Microsoft office files constitute rough clusters in the embedding space. The specific categories we selected were either constituting a particular file type from the "file path" field (e.g., ".pyc" files or ".dll" files), or types of vocabulary words from different fields that almost always appear together (e.g., registry keys and registry values). While this is merely a heuristic, we can clearly see that along these categories, clusters do emerge among like points. For example, registry keys and values are generally embedded near other registry keys, and values and are generally further away from embeddings of words in other categories. While not perfect, the training paradigm clearly generates meaningful embeddings based on the contexts in which they appear, similar to the nonfederated model featured in our previous work [12].

Results for sysclient 0203-0205

While the model was trained on data from these hosts, the model had no access to the red team activities performed on sysclient 0203 and sysclient 0205 during training. Since 0204 is an unattacked host with its data in the training corpus, we consider this to be our "control" host. Overall, the federated model is able to distinguish between benign activities from red team activities on 0203 and 0205 and generalizes well to the test set on 0204.

As seen in figure 4 (page 34), the anomaly score distributions for benign and red team activities on both attacked hosts are such that there is a clear separation between a large portion of the malicious activity, and the majority of the remaining activity. The scores for sysclient 0204 are similar to the benign scores of the other two hosts.

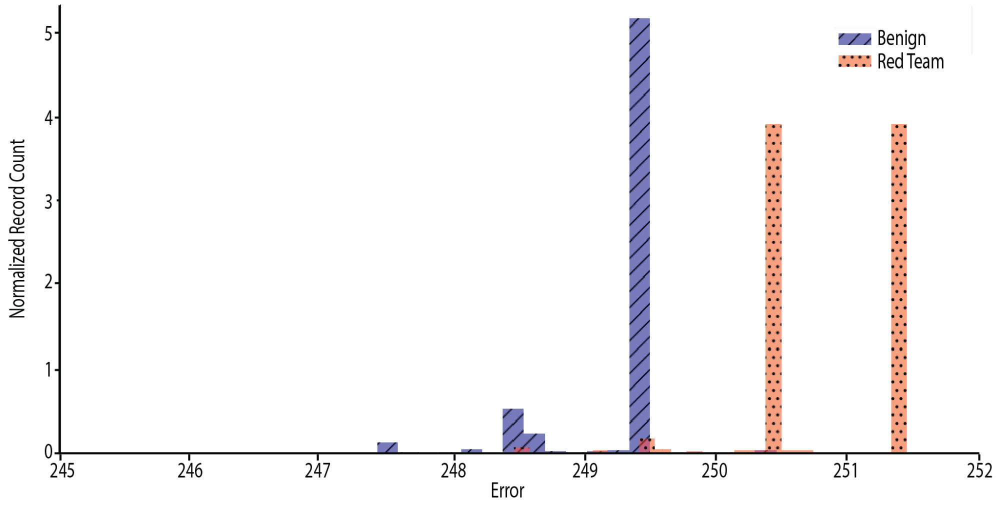
Emulating deployment of this model in a Security Operations Center (SOC), we opted for a temporally based anomaly detection system. To that end, we gather our test data into 60-minute buckets, calculate the percentage of records in each bucket over the 99.9th percentile of a random sample of benign data from the given host during the training period and display the resulting time series in figure 6 (page 36). Generally speaking, anomaly scores are within the expected levels outside the attacks on sysclient 0203 and sysclient 0205 and remain at consistently elevated levels during the times each were attacked. There were some exceptions, however. The spikes in our metrics at t=1500 and t=2880 are correlated with large spikes in the volume of WMI (Windows Management Instrumentation) activity on each host. These spikes are consistent with our previous work and we, also as before, have elected to treat these spikes as nonmalicious anomalies. There are some red team events that appear at t=1000, which did not appear in our previous work. This is most likely due to an update in the labeling procedure we employed. That the anomaly scores do not spike around this time is not surprising given the fact that there are only around 100 red team events in total.

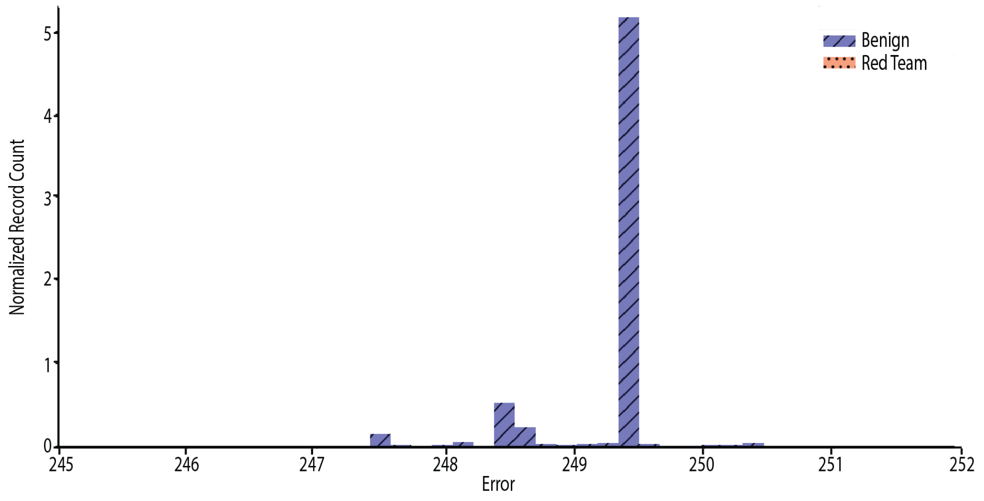
For sysclient 0204, anomaly scores appear to be within expectations and in this SOC scenario, no alerts would be generated. Again, this is exactly what one would want to happen, since the red team did not attack this host.

Results for sysclient 0201 and 0202

This is a challenging scenario for the model. Recall that the training data for the model included no records from sysclient 0201 nor sysclient 0202, and were chosen because 0201 contained activities from the red team constituting multiple stages of an attack and 0202 did not.

As we see in figure 7 (page 37), the results for sysclient 0201 are decent, especially given that the training of the FL model had not seen data from this host during training. Here the separation between the anomaly score distributions by class are not as clear, indicating that either there were high volumes of benign activities which were different enough from the training data that they were difficult to reconstruct, or high volumes of the red team activities were too similar to training data. This also means that precision and recall for a naive classifier using the anomaly score as a threshold had significantly degraded performance. However, using the scores





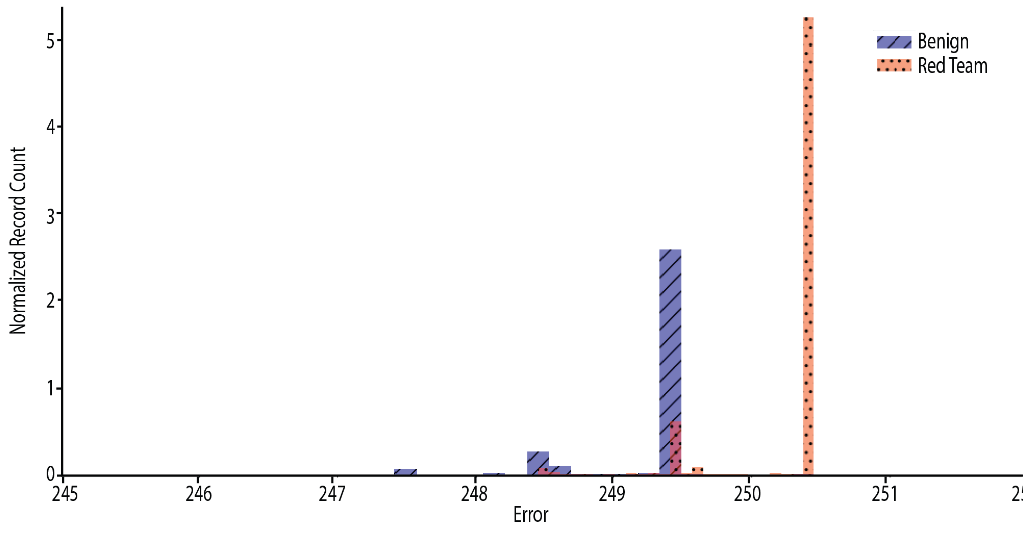
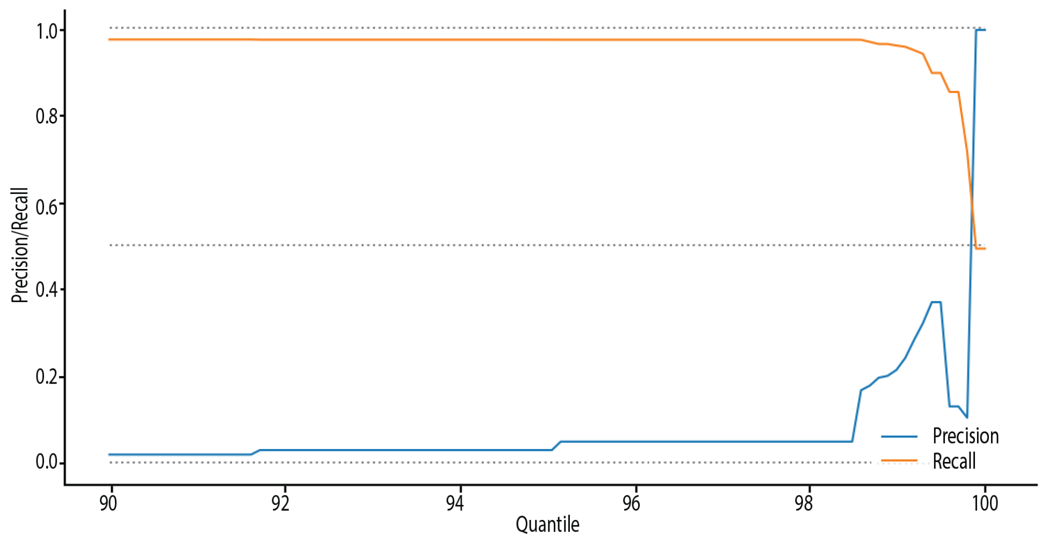


FIGURE 4. These histograms of anomaly score by class (benign and red team) on sysclient 0203 (top) sysclient 0204 (middle) and sysclient 0205 (bottom) show a clear separation between a large portion of the malicious activity, and the majority of the remaining activity. The scores for sysclient 0204 are similar to the benign scores of the other two hosts.



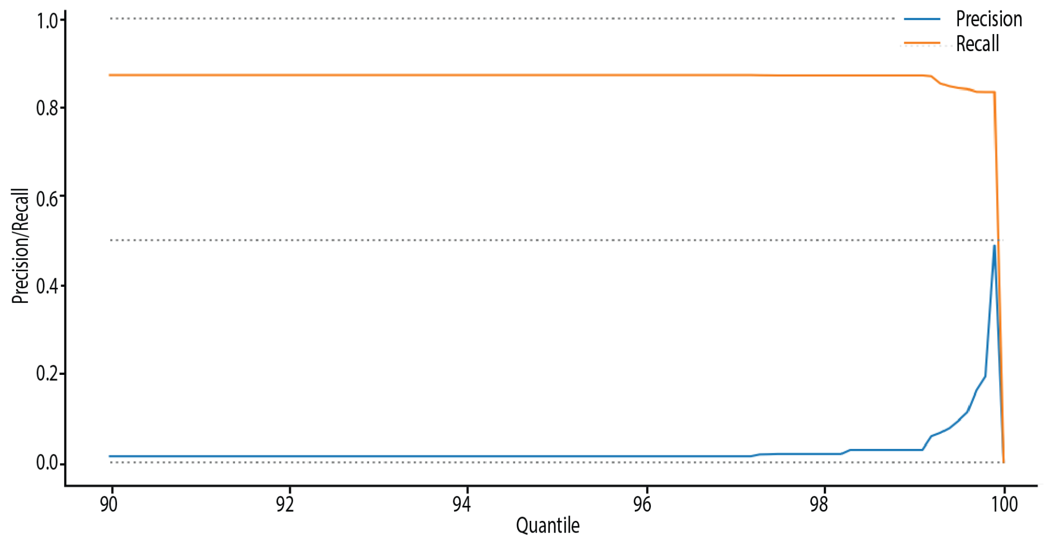


FIGURE 5. For these precision and recall curves for sysclient 0203 (top) and sysclient 0205 (bottom), we used a threshold to create a naïve classifier on the test data. For each quantile, q, between the 95th and 100th percentile of our test loss distribution, we classify every record with a loss above q as being anomalous.

as a temporal anomaly detection scheme, we see that the model does generalize somewhat. Anomaly score spikes early on match the high volumes of red team activities. Spikes around t=1500 and t=2880 are consistent with sysclients 0203-0205, and lend further credence to the notion that these are benign, but anomalous events.

The results for sysclient 0202 are strong. The anomaly score distribution is very similar to the benign activities of the other hosts. Additionally, these scores remain within expectations in the temporal-based anomaly detection scenario. All told, this very clearly demonstrates that the FL model generalizes well.

Discussion

While we do not perfectly mirror our results from the previous work on sysclient 0201-0204, we do demonstrate that such a FL paradigm has the potential to aid in real-time cyber threat hunting. The results for sysclient 0201 do highlight the need for the training data FL model to be sufficiently close to data in the wild. The degradation in performance as compared to our previous work is most likely due to the model having not seen enough data that is similar to the data that characterizes normal activities on 0201.

Another key component of transitioning such a technique to practice is determining the appropriate

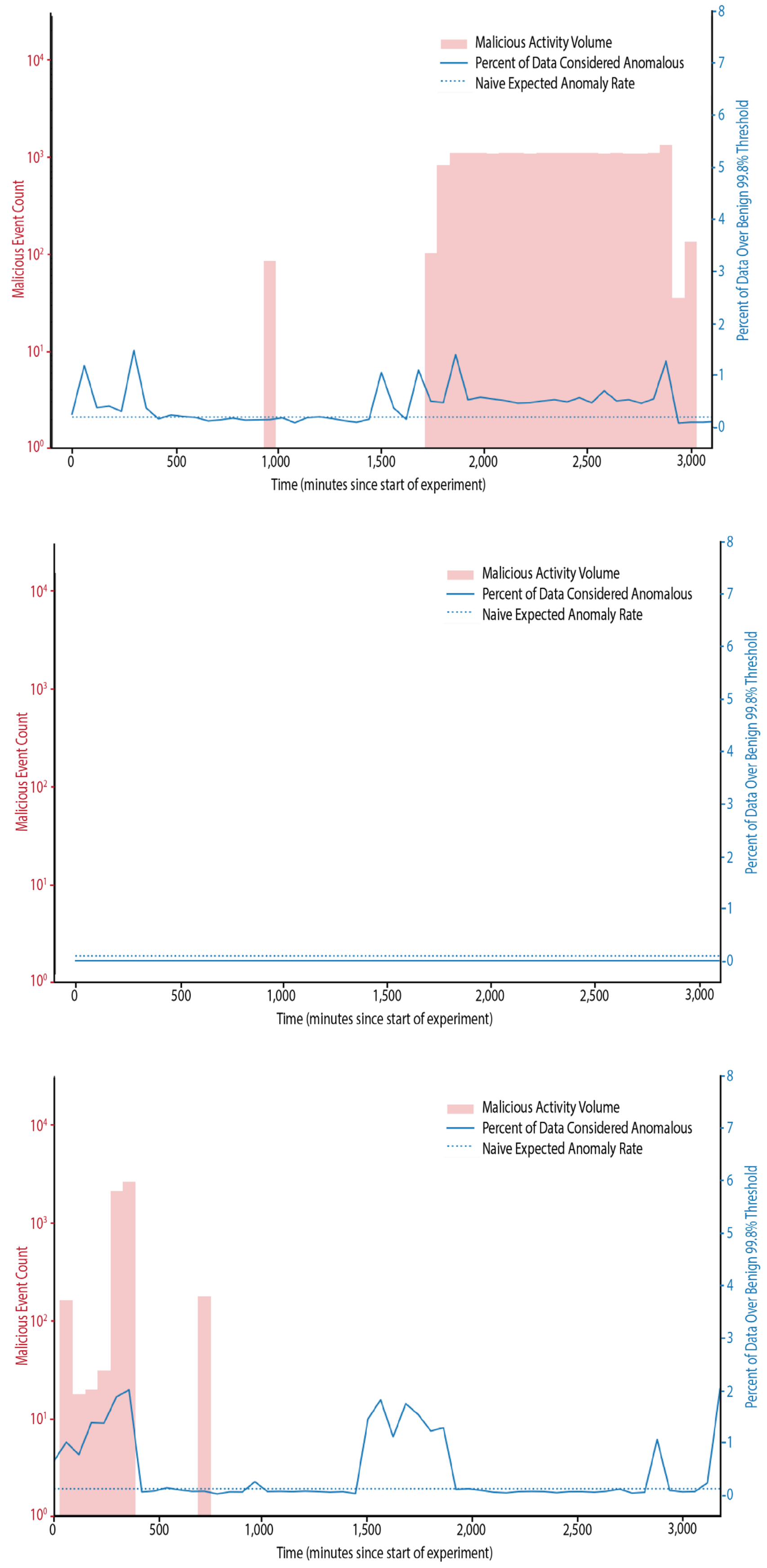


FIGURE 6. In this comparison of the amount of red team activity in a given 60-minute bucket, and the amount of activity over the 99.5th percentile of the error distribution for the training data on the associated host,the dashed line is set at the average expected level of the graph, 0.5 percent. This figure includes two attacked hosts, sysclient 0203 (top) and sysclient 0205 (bottom), as well as the unattacked host, sysclient 0204 (middle).

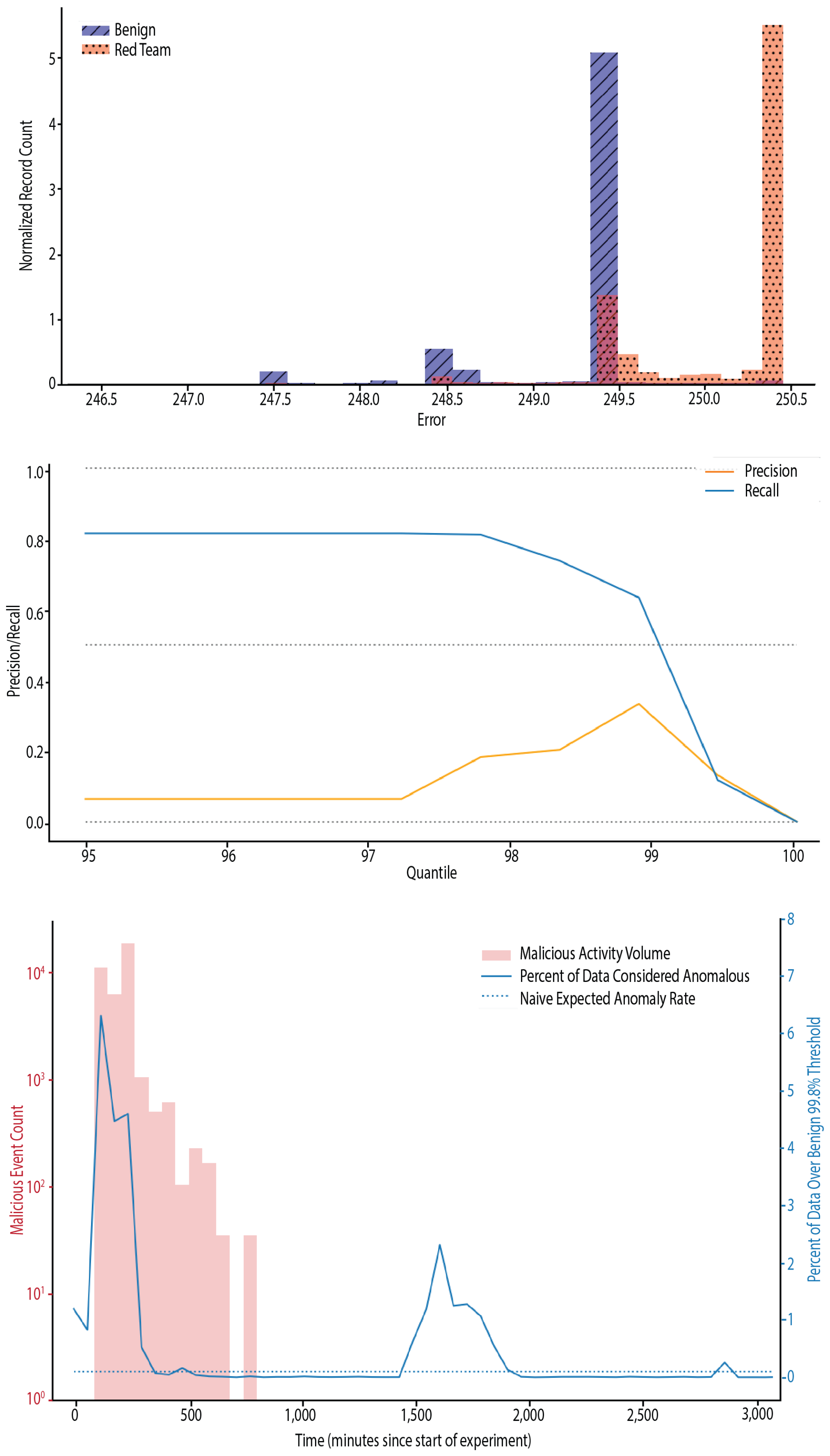


FIGURE 7. In these results for sysclient 0201, the separation between the anomaly score distributions by class are not as clear; however, using the scores as a temporal anomaly detection scheme, we see that the model does generalize somewhat. Anomaly score spikes early on match the high volumes of red team activities.

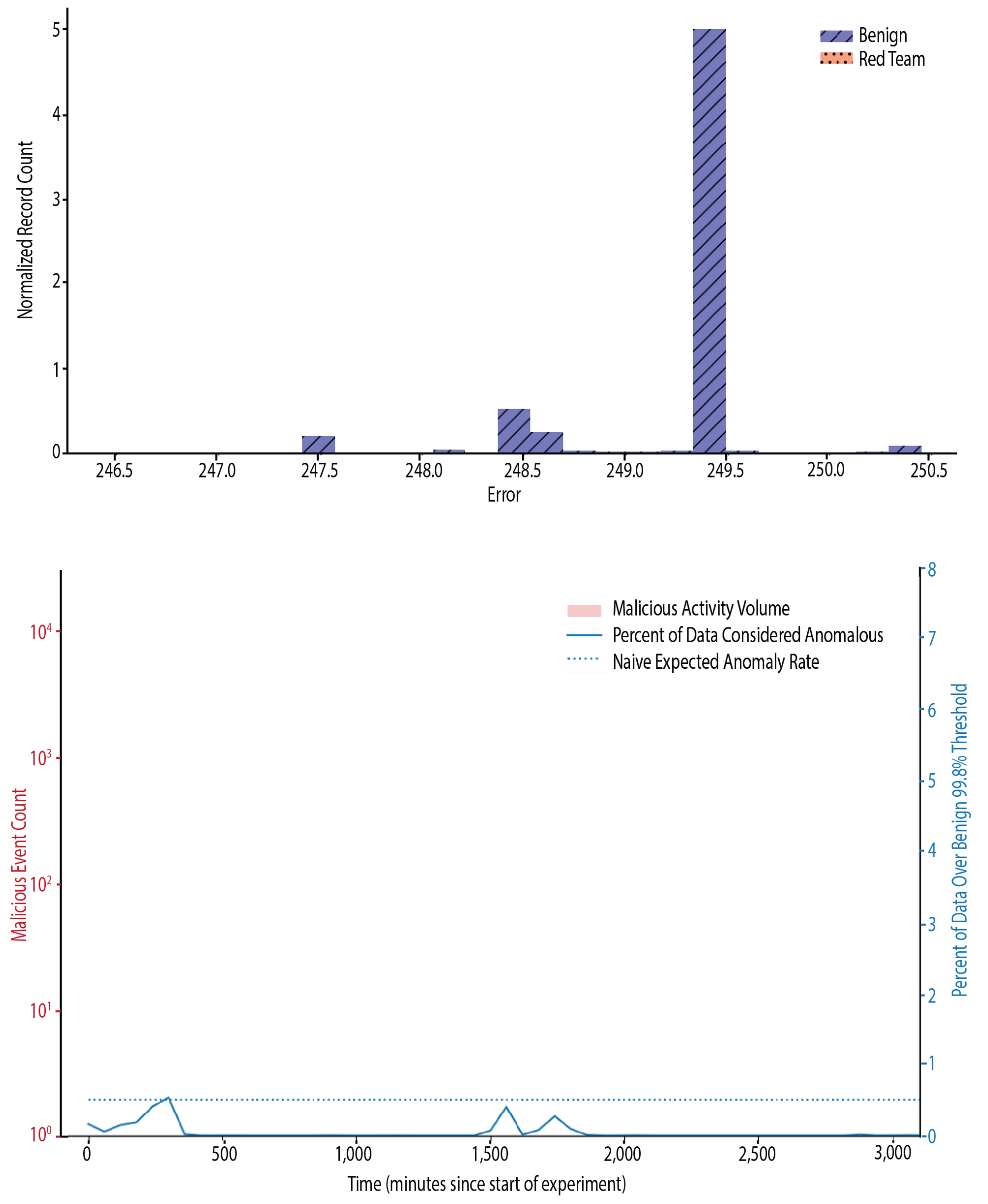


FIGURE 8. Results for sysclient 0202 shows the anomaly score distribution is very similar to the benign activities of the other hosts.

hyper parameters. In our experiments, we set many of the hyperparameters to be consistent with our previous work. We do so in order to enable an apples-to-apples comparison of our works. In practice, one should expect the depth of the network, the size of the code layer, and even the dimensions of the vocabulary embeddings to scale with the complexity of the underlying data. In the case of training an FL model on orders of magnitude more hosts than ours, we advise to perform an ablation study.

Future work

The clearest future work would be to attempt to scale this FL paradigm to even more hosts on OpTC. Such an endeavor would be extremely informative for those wishing to employ scalable AI/ML-enhanced cybersecurity analytics in an SOC. Future work also includes a deeper exploration finding suitable hardware and software that can bring the speed of training a FL model on par with a non-FL one (especially for the scalability concerns). The tensorflow distributed learning module includes support for a strategy that is optimized for tensor processing units (TPUs). Testing out this concept on such architecture would be an interesting endeavor.

As we have alluded, privacy-preserving methodologies, if perfected, could enable cross-organizational model training and collaboration for enhanced cybersecurity capabilities without being prohibitively risky. Recent literature does demonstrate that differential privacy can be employed in deep learning models, but is very difficult. Research endeavors in this space are needed, in order to ensure that attackers cannot continue to take advantage of the lack of collaboration among defenders across organizations.

References

[1] Hutchins E, Cloppert M, Amin R. "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains." *Leading Issues in Information Warfare & Security Research 1.* 2011. Available at: https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf.

[2] Strom BE, Applebaum A, Miller DP, Nickels KC, Pennington AG, Thomas CB. "Mitre att&ck: Design and philosophy." 2018 July, revised 2020 March. MITRE Corporation technical report MP180360R1.

[3] Mossburg E, Fancher JD, Gelinne J. "The hidden costs of an IP breach, cyber theft and the loss of intellectual property." D*eloitte Review 19*. 2016: 106–121.

[4] Snort webpage. Available at: https://www.snort.org/.

[5] Sigma GitHub webpage. Available at: https://github.com/SigmaHQ/sigma.

[6] Goodfellow I, Bengio Y, Courville A. *Deep Learning.* MIT Press; 2016. ISBN: 9780262035613.

[7] Sakurada M, Yairi Y. "Anomaly detection using autoencoders with nonlinear dimensionality reduction." In: *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis (MLSDA'14);* 2014; Association for Computing Machinery, New York, NY: pp. 4–11. Available at: https://doi.org/10.1145/2689746.2689747.

[8] Hawkins S, He H, Williams G, Baxter R. "Outlier detection using replicator neural networks." In: Kambayashi Y, Winiwarter W, Arikawa M, editors. *Data Warehousing and Knowledge Discovery DaWaK;* 2002. Lecture Notes in Computer Science, vol 2454. Springer, Berlin, Heidelberg. Available at: https://doi.org/10.1007/3-540-46145-0_17.

[9] Wong V, Emanuello J. "Robustness of ML-enhanced IDS to stealthy adversaries." In: *AI/ML for Cybersecurity: Challenges, Solutions, and Novel Ideas at SIAM Data Mining;* 2021. Available at: https://doi.org/10.48550/arXiv.2104.10742.

[10] Ramström K. "Botnet detection on flow data using the reconstruction error from Autoencoders trained on Word2Vec network embeddings." PhD thesis. Uppsala Universitet, 2019.

[11] Liu F, Wen Y, Zhang D, Jiang X, Xing X, Meng D. 2019. "Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise." In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19);* 2019; Association for Computing Machinery, New York, NY: pp. 1777–1794. Available at: https://doi.org/10.1145/3319535.3363224.

[12] Golczynski A, Emanuello JA. "End-To-end anomaly detection for identifying malicious cyber behavior through NLP-based log embeddings." In: *Proceedings of the First International Workshop on Adaptive Cyber Defense;* 2021. ArXiv: abs/2108.12276.

[13] Kairouz P, McMahan HB, Avent B, Bellet A, Bennis M, Bhagoji AN, Bonawitz K, et al. "Advances and open problems in federated learning." *Foundations and Trends in Machine Learning.* 2021;14(1–2):1–210.

[14] Ghimire B, Rawat DB. "Recent advances on federated learning for cybersecurity and cybersecurity for federated learning for Internet of Things." *IEEE Internet of Things Journal.* 2022;9(11):8229–8249. doi: 10.1109/JIOT.2022.3150363.

[15] Zhou Y, Wu J, Wang H, He J. "Adversarial robustness through bias variance decomposition: A new perspective for federated learning. In: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management (CIKM '22);* 2022; Association for Computing Machinery, New York, NY: pp. 2753–2762. Available at: https://doi.org/10.1145/3511808.3557232.

[16] Boenisch F, Dziedzic A, Schuster R, Shamsabadi AS, Shumailov I, Papernot N. "When the curious abandon honesty: Federated learning is not private." In: *2023 IEEE Eigth European Symposium on Security and Privacy (EuroS&P);* 2023; Delft, Netherlands: pp. 175–199. doi: 10.1109/EuroSP57164.2023.00020.

[17] DARPA. Operationally Transparent Cyber (OpTC) Data Release, 2019. Available at: https://github.com/FiveDirections/OpTC-data.

[18] Anjum MM, Iqbal S, Hamelin B. 2021. "Analyzing the usefulness of the DARPA OpTC dataset in cyber threat detection research." In: *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies (SACMAT '21);* 2021; Association for Computing Machinery, New York, NY: pp. 27–32. Available at: https://doi.org/10.1145/3450569.3463573.

[19] The MITRE Corporation. "Data Model." MITRE Cyber Analytics Repository. Available at: https://car.mitre.org/data_model/.

[20] Denning DE. "An intrusion-detection model." *IEEE Transactions on Software Engineering.* 1987;13(2):222–232. doi: 10.1109/TSE.1987.232894.

AI/ML Anomaly Detection Analytics for Wireless Internet of Things Systems

Stephanie Polczynski, Samuel Fox, Erik Brasile, NSA
Dr. Robert Simon, George Mason University

[Photo credit: iStock.com/NatalyaBurova]

Around the world, critical infrastructures increasingly incorporate Internet-of-Things (IoT) devices and networks. Since, by definition, most IoT devices are connected to national if not global networks, the explosive growth of IoT systems introduces a vast new set of potential vulnerabilities and exploitable cyber threat vectors. IoT cyberattack classes range from injecting false or misleading sensor data and distributed denial-of-service attacks to long-term stealthy advanced persistent threats (APTs).

This article describes our efforts to provide tools to detect potentially compromised IoT devices based on behavioral analysis of their local wireless [e.g., Bluetooth, Bluetooth Low Energy (BLE), ZigBee, etc.] communications. We have integrated these tools into a project called the Wireless IoT Monitoring System (WIMS). We adopt a classification-based machine learning (ML) approach, whereby device-based IoT packet exchanges are encoded into a set of images. Our ML models are trained to recognize well-behaving devices and identify anomalous traffic through these generated images as an indication of a potentially compromised device or system.

The wireless IoT protocol landscape consists of dozens of over-the-air protocols based on both open-source and proprietary standards. While many types of attacks against older protocols such as Bluetooth have been identified, attacks against newer protocols such as long-range wide area network (LoRaWAN) and narrowband (NB)-IoT are less understood and the threat space and exploit scenarios are not as well-defined. To address this discrepancy, we leverage transfer learning (TL) techniques. Using TL, we can train an anomaly detection classifier for a well-known class of attacks, say against Bluetooth, and reuse that knowledge to shorten the training time and improve the accuracy of anomaly detection for a newer protocol, such as LoRaWAN.

We first provide a brief background into the wireless IoT threatscape, and then discuss our IoT monitoring system. Next, we describe how we transform IoT packet captures into picture images representing encoded time series data from the network traffic and use ML on those images to identify anomalous activity. Thereafter, we explain how we use TL to rapidly train image classifiers on new protocols and attack classes. Lastly, we provide some experimental results and offer conclusions.

Threatscape for wireless IoT

There has been limited work on using ML to monitor wireless IoT traffic and detect suspicious behavior. Current solutions primarily focus on wired or 802.11 wireless IoT devices and focus on the communication patterns between an IoT device and the host server. For example, one might monitor the encrypted web communications of a smart home hub/gateway from the router to the manufacturer's cloud server. These solutions would examine features like the number of times the device usually communicates with the server, average data transmission size, frequency of communications, etc. However, these solutions are not examining the local wireless communications of the IoT devices. IoT end devices often use protocols like Bluetooth, ZigBee, Z-Wave, Thread, NB-IoT, LoRaWAN, and now 5G to communicate with the managing device, hub, gateway, or cloud server.

Monitoring traffic over these protocols requires special equipment that is often designed for use by radio-frequency (RF) experts and is not tailored for use by the average network analyst and certainly not designed to be used for long-term monitoring and detection. Further, IoT devices typically lack traditional security mechanisms such as firewalls, antivirus programs, system logs, etc. that can be used to prevent, detect, and analyze potential malicious activity. Because of this, we sought to develop a wireless IoT monitoring system that performs anomaly detection using what is often the only source of information available from wireless IoT devices—the packets they transmit to communicate between other IoT end devices and IoT gateways.

To fully grasp the IoT problem set, it is useful to understand the scale of the IoT space from the number of protocols available and in use by technology developers, to the complex networks spanning multiple protocols. The number of IoT protocols is a relatively difficult number to pin down. A quick Internet search of "How many IoT protocols are there?" yields results of anywhere from the top six IoT protocols to 26 IoT protocols [1]. Most of these articles also acknowledge that their lists are "extensive—but not exhaustive." As wireless technologies improve, new protocols are constantly being developed to cater to the specific needs of mobile networks. The Fifth Generation of Mobile Telephony (5G) standard, for example, was "functionally frozen in June 2018 and fully specified by September 2019." 5G promises support for a "massive Internet of Things" integrating "the operational aspects that apply to the wide range of IoT devices and services anticipated in the 5G timeframe" [2]. On the other hand, some IoT sensor devices have been in use for decades and will continue to be so for many more years, without the opportunity for updates. Therefore, IoT-enabled devices often have support for not only new technologies, but also legacy protocols.

In addition to the number individual protocols available for use, many modern IoT networks make use of several IoT protocols to provide better quality of service (QoS) to their users. Amazon Sidewalk, for example, uses (at a minimum) BLE [2.4 gigahertz (GHz) industrial, scientific, and medical (ISM) band] for short-range communications and LoRa [900 megahertz (MHz) ISM band] for long range [3]. Many devices support multiple methods of communication and automatically switch between the protocols based on any number of metrics determined by the device manufacturer. Each protocol a device supports is another communication link and therefore potential attack vector for malicious actors.

How does an organization protect itself when incorporating devices with IoT interfaces into its operations? Simply not using devices with IoT capabilities is likely the safest option; however, this option is growing increasingly difficult and impractical. It is not uncommon for device manufacturers to include one or more wireless protocol interfaces. There are many reasons to do so from a manufacturer's perspective to include, QoS (as discussed previously), redundant links for firmware updates, mesh network support, device metrics, information collection, etc.

Wireless IoT monitoring options

Before developing analytics to perform behavioral analysis of wireless IoT traffic, a process first needed to be developed to collect the traffic from the IoT devices. There are two primary means of doing this currently. The first is using commercially available IoT protocol analyzers/sniffers, which automatically detect and demodulate supported IoT signals and output the observed traffic in an easily parseable, often packetized format for easy analysis of the data. The second method is using software-defined radios (SDRs) which provide a generic radio front end that captures raw samples of the RF spectrum. These devices allow for adjustable tuning and instantaneous bandwidth to cover nearly any portion of the RF spectrum; however, they require specialized expertise to operate to obtain the IoT data.

IoT protocol analyzers

Commercially available protocol analyzers/packet sniffers are available to monitor the most common wireless IoT protocols to include Bluetooth Classic [basic rate/enhanced data rate (BR/EDR)], BLE, Wi-Fi, and 802.15.4-based protocols such as Zigbee and Thread [4, 5, 6]. Many implementations are hardware-based solutions that cater to specific frequency bands or specific protocols. Higher-end wireless sniffers can often detect packets from several different protocols, perform simultaneous collection on each of them, while also providing the raw RF data stream. Example solutions include the Spanalytics PANalyzer, the Ellisys Vanguard, and Frontline X500, all of which target the 2.4 GHz ISM band.

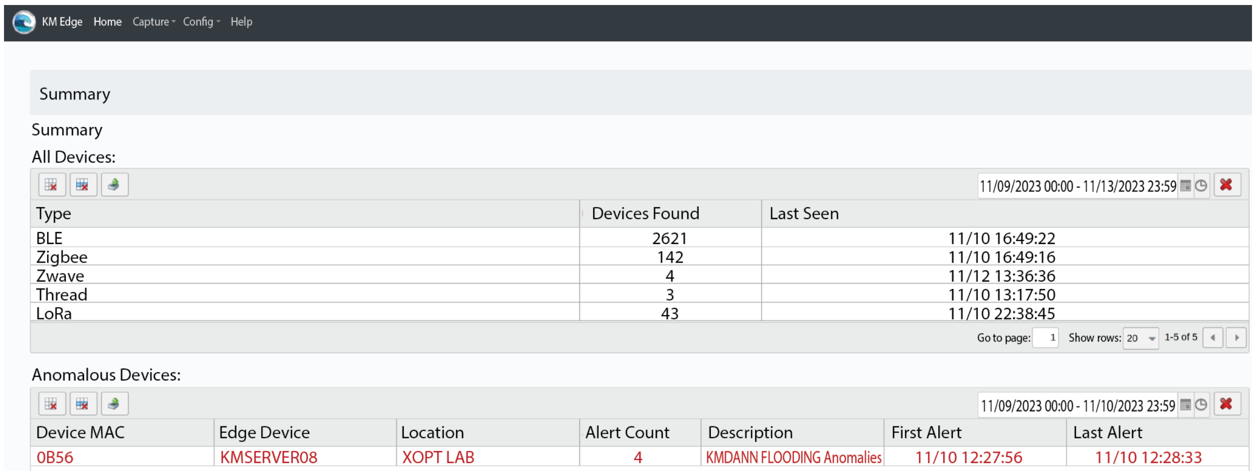
The difficulty of dealing with hardware-based RF solutions is that in order to adapt to new protocols or protocols in a different frequency band, additional hardware or at a minimum new firmware/drivers must be pushed to all devices. Adding new hardware introduces many considerations into a system's design to include physical footprint, complexity (e.g., potentially multiple manufacturers/interfaces), power consumption, additional cost, maintenance, etc.

These commercial-off-the-shelf (COTS) packet sniffers are a great option for targeting the most common protocols being used. With that being said, because of the vast number of protocols in the IoT space, quite a few are not supported or require additional hardware to cover. Each of the manufacturers previously listed provide additional hardware expansions to provide support for additional popular protocols in the 900 MHz ISM band such as LoRa and Z-Wave. This however still leaves quite a few protocols unsupported and again raises the issues of adding additional hardware to the system architecture.

Software-defined radios

To attempt to fill in the gaps and solve some of the hardware scalability issues, software-defined radios (SDRs) provide flexibility and adaptability. SDRs provide a "generic" radio front end that captures raw samples of the RF spectrum. These devices allow for adjustable tuning and instantaneous bandwidth to cover nearly any portion of the RF spectrum needed. Many provide onboard field-programmable gate arrays (FPGAs) for use as signal processors and provide high-speed interfaces such as universal serial bus (USB), Ethernet, peripheral component interconnect express (PCIe), etc. for offloading to a separate processor. These separate processors can make use of modern processor units [central processing units (CPUs), graphics processing units (GPUs), etc.] to process wideband captures of the spectrum in real time. Because the SDR front end is relatively generic, the system used to process the RF sample stream can be upgraded to meet the needs of the signal processing toolchain. Additionally, the architecture of SDR-based systems makes processing multiple protocols in parallel relatively simple.

Because the radio front-end interface is abstracted from the processing unit, developing signal packet sniffers becomes extremely modular and software based. Signal processing toolchains can be written in any language supported by the processing unit which is likely running a modern operating system and can support almost any compiler. GNU Radio, for example, is a popular open-source SDR development framework commonly used by hobbyists and researchers to implement signal processing toolchains.



Summary

Summary

All Devices:

11/09/2023 00:00 - 11/13/2023 23:59

Type	Devices Found	Last Seen
BLE	2621	11/10 16:49:22
Zigbee	142	11/10 16:49:16
Zwave	4	11/12 13:36:36
Thread	3	11/10 13:17:50
LoRa	43	11/10 22:38:45

Go to page: 1 Show rows: 20 1-5 of 5

Anomalous Devices:

11/09/2023 00:00 - 11/10/2023 23:59

Device MAC	Edge Device	Location	Alert Count	Description	First Alert	Last Alert
0B56	KMSERVER08	XOPT LAB	4	KMDANN FLOODING Anomalies	11/10 12:27:56	11/10 12:28:33

FIGURE 1. This Wireless IoT Monitoring System (WIMS) dashboard provides an overview of which IoT protocols and devices were observed in the coverage area. It also displays alerts for devices with detected anomalous traffic.

GNU Radio is built on C/C++ and Python and greatly simplifies the work required to develop an SDR protocol sniffer implementation. This provides an additional benefit due to the availability of open-source implementations of nearly every IoT protocol. Adding support for a protocol could be as simple as installing an open-source GNU Radio module.

To further increase modularity, a management framework could be used to abstract the front-end radios from any signal processing units. A management framework would control distributed antennas as well as multiple SDRs monitoring multiple portions of the spectrum in different locations simultaneously. This management framework would be able to control not only the portions of the spectrum being monitored but also control the protocols being processed. An SDR solution is a major improvement over an IoT protocol analyzer since it is extremely modular and adaptable to the specific requirements of an organization.

Distributed monitoring architecture

For the initial effort, we utilized commercial IoT protocol analyzers/sniffers due to their ease of use and support for the most common wireless IoT protocols (e.g., Bluetooth, BLE, ZigBee, etc.). However, as we continue to add support for additional wireless IoT protocols to the system which do not have commercially available sniffing tools, we are also adding SDR support to enable the processing of further protocols.

To enable the use of artificial intelligence (AI)/ML anomaly detection analytics for wireless IoT traffic, we designed a distributed architecture and the supporting software to monitor the wireless IoT traffic over an area. The area to be monitored could be a room, building, campus, etc. if sufficient IoT traffic sniffers were placed throughout the area to ensure adequate coverage to collect the wireless IoT traffic. The sniffers need to be distributed optimally through the space due to low power and low-transmission range of most wireless IoT protocols. These distributed sniffers will send all observed wireless IoT traffic to a central server for further processing. This system is called the Wireless IoT Monitoring System (WIMS).

The central WIMS server provides two primary functions:

1. **Situational awareness into which wireless IoT protocols and devices are observed in the coverage area.** This includes a count of how many devices of each protocol were observed over a set time period, the addresses of the devices, and the sniffer(s) that detected the device to aid in geolocation. Additionally, if the end device uses a resolvable public media access control (MAC) address—the system will provide as much identifying information on the device as possible.
2. **ML-based anomaly detection analytics on the observed IoT traffic.** If a potential anomaly is flagged, it will be displayed on the alert dashboard, can be clicked-into for further information, and alerts can optionally be forwarded to other security tools or to designated security officials.

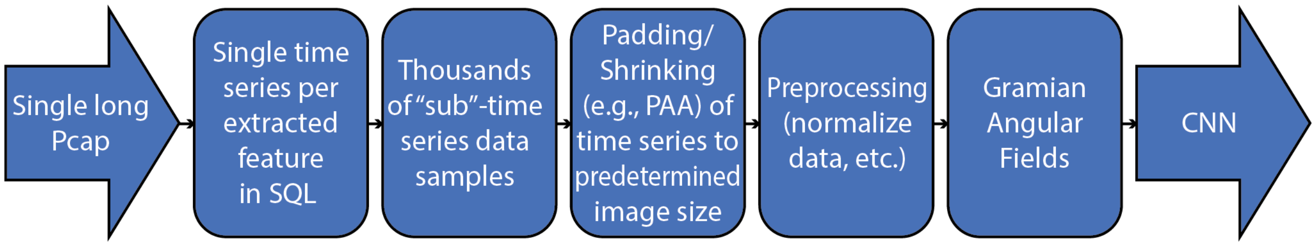

Single long Pcap
Single time series per extracted feature in SQL
Thousands of "sub"-time series data samples
Padding/ Shrinking (e.g., PAA) of time series to predetermined image size
Preprocessing (normalize data, etc.)
Gramian Angular Fields
CNN

FIGURE 2. In this ML processing pipeline, the IoT data is ingested in PCAP form, converted into a time series for each feature, and then pre-processed prior to being transformed into a Gramian angular field (GAF) picture and inputted into a CNN for anomaly detection.

Anomaly detection approach

Wireless behavioral monitoring systems such as WIMS must ingest and process large numbers of IoT packet data streams. Sophisticated IoT cyberattacks may only be detectable by understanding the statistical relationships both within each packet and between packets in each data stream. Given the sheer number of packets and complex interrelationships between packets inside of data streams, current ML approaches are particularly attractive since they automatically learn to classify individual datasets into discrete classes.

IoT data streams are a type of time series information. These time series form data types that can be viewed as one-dimensional or two-dimensional grids. Convolutional neural networks (CNNs) do extremely well at classifying grids. For instance, CNNs are known to excel at image classification, where each image is processed as a two-dimensional grid of pixel values. By treating IoT data streams as time series grids representing packet data, WIMS takes advantage of existing computer vision and image processing ML capabilities.

The fundamental ML approach used is to turn packet captures from a particular IoT protocol into a set of images which are then analyzed by a CNN. WIMS uses a Gramian angular field (GAF), described below, to convert the time series into a two-dimensional grid which can be treated as an image.[a] The advantage of this approach is that it preserves temporal dependencies among packet events, provided the events are properly encoded. The CNN is trained to recognize anomalous versus normal traffic in a process akin to image classification. Once a stable model is developed, that model is used for TL to a new protocol.

After suitably accurate results for correct classification (i.e., normal versus anomalous) are achieved with this method, the next goal is to transfer the well-trained CNN for anomaly detection on other protocols. When building this capability, we first sought to demonstrate, with nearly 100 percent accuracy, the ability to detect a Bluetooth response flooding attack using a CNN image-classification algorithm, and then use TL to substantially reduce the training time for anomaly detection of a response to a flooding attack in an 802.15.4 network such as ZigBee.

As traffic is collected from the IoT traffic sniffers and forwarded to the central server, the data is stored in a MySQL database in the form of packet capture (PCAP) files. The first steps of the ML processing pipeline are to turn each individual PCAP into an image which can be processed by a CNN. This pipeline is shown below and is explained in detail in the following section.

Image production

The first step in transforming packet captures into images is to extract time series data from the database from each feature of interest. Example features of interest are packet interarrival time, packet size, and protocol message type. Specifically, each value in the time series comes from one packet. For the database used in the WIMS, this corresponds to one row. The time series is split into fixed size chunks corresponding to a certain duration of elapsed real time. The current default is one second. This produces a subseries for each feature.

When processing data extracted from a database (e.g., training, validation, testing) or a dictionary (i.e., real-time prediction), the values of each feature must be parsed in an appropriate fashion. The parser

a. It should be noted that the images produced by WIMS cannot be readily interpreted by human analysts.

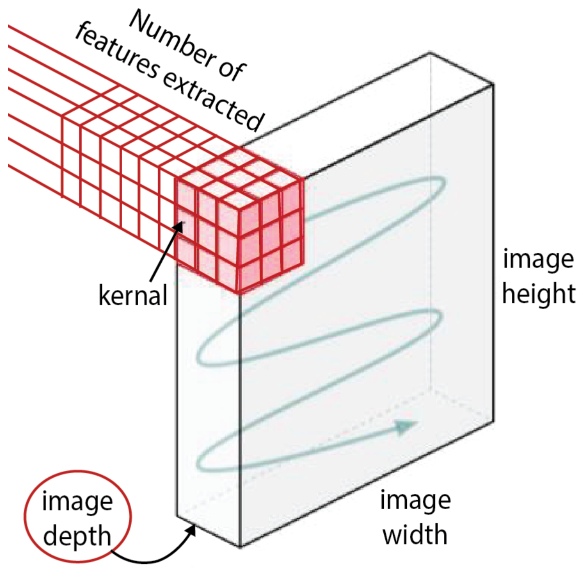


FIGURE 3. In building this Gramian angular field (GAF) representation, the value of each feature in the time series is converted into a polar coordinate system, and a two-dimensional grid (i.e., the image) is produced where each entry is the trigonometric sum between each pair of points.

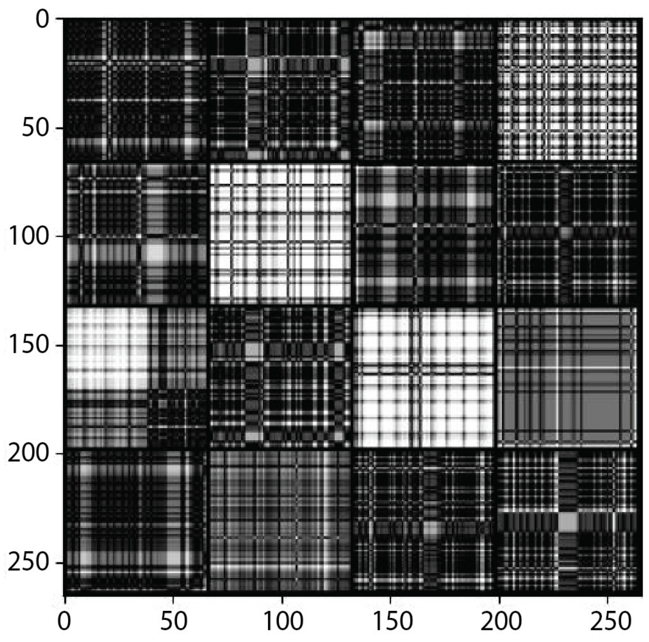


FIGURE 4. This sample Bluetooth GAF representation contains 16 separate subsequences from a Bluetooth flooding attack. The initial input was packet interarrival time.

replaces any NULL values from the database with a chosen pad value for the feature, then converts the data extracted from the database with values of the appropriate data type. The rules for pad values are as follows:

1. It cannot be a value which could legitimately arise from a true observation; for example, don't pick a positive value for packet length, for example.
2. It makes sense in the context of its respective feature.
3. It can be represented by the chosen data type for this feature; for instance, don't pick a negative pad value when using an unsigned integer like np.uint32.

At this point a Gramian angular field is produced for each subseries. This is done by rescaling all values to fall within the interval [0,1]. These values are then put into a polar coordinate system by encoding the value as the angular cosine and the time stamp as the radius. A square matrix is then used where each entry is the trigonometric sum between each pair of points. This enables the identification of temporal correlations between different time intervals. A square Gramian matrix is produced which is then transformed into an red-green-blue (RGB) image.

A key part of the data processing pipeline is to use stackable GAF representations, each corresponding to the same time slice for each feature to form a tensor of arbitrary dimensionality. This allows for an N-channel image that allows a network traffic representation of arbitrary complexity, providing maximum flexibility for feature selection. This is shown in the figure 3.

Figure 4 shows an example image, consisting of 16 samples, from a training dataset used to identify a Bluetooth flooding attack. The image shown here is for interarrival time. The advantage of this approach is that we can make full use of existing ML techniques for image classification for anomaly detection. We train our CNN to function as a one-class classifier. It learns to recognize "normal" images that correspond to non-anomalous traffic and to reject non-normal images. These rejected images correspond to anomalous traffic.

Convolutional neural network and transfer learning

The WIMS treats anomaly detection as a binary classification problem. The approach is to produce

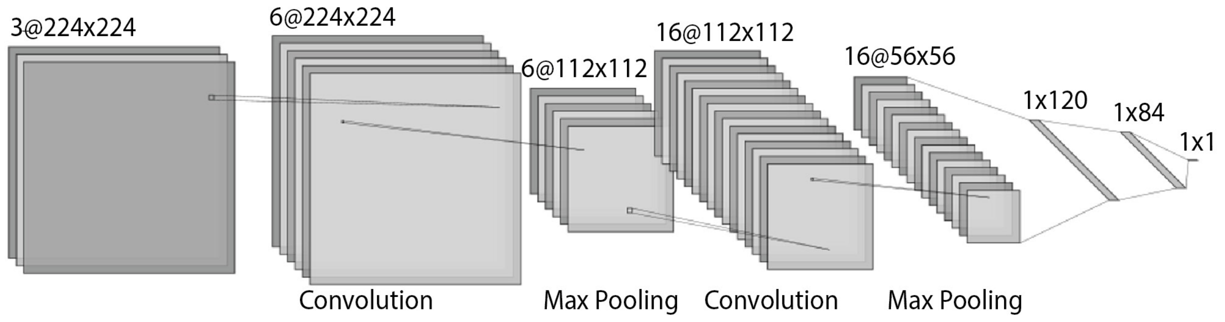


FIGURE 5. In this WIMS CNN, the CNN uses two convolutional layers with a relatively low number of output channels, three fully connected dense layers with a relatively low number of parameters, max polling layers, and ReLU activation.

images using the technique just described of both normal behaving traffic and anomalous traffic. The CNN is then trained to predict whether a newly presented sample falls in the normal class of images it has been trained on. All samples not classified as normal are considered anomalous and can generate an alert.

The WIMS uses a relatively small CNN. It is shown in figure 5. It uses two convolutional layers with a relatively low number of output channels, three fully connected dense layers with a relatively low number of parameters, max polling layers, and rectified linear (ReLU) activation. Taken together there are approximately six million trainable parameters. Considerable effort was put into fine tuning the CNN values to produce highly accurate results.

Let TP represent the true positive rate, TF represent the true negative rate, FP represent the false positive rate, and FN represent the false negative rate. We evaluated the performance of our approach using F1 and the Matthews correlation coefficient (MCC) metrics. An F1 score is the harmonic mean of precision and recall. The F1 score can be written as

$$F1 = \frac{2\text{*TP}}{(2 * \text{TP})+\text{FP} + \text{FN}}.$$

For binary classifiers, MCC shows the correlation between predictions and actual observations. A value of 1.0 shows perfect prediction, 0 is essentially random, while -1.0 represents completely incorrect predictions. The MCC can be calculated directly from the 2x2 confusion matrix by calculating the following:

$$MCC = \frac{(TP * TN) - (FP * FN)}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}}.$$

In our work, once routine F1 and MCC scores of 0.98 or above were obtained, it was possible to start the process of TL. The goal was to take a CNN trained for Bluetooth anomaly detection and transfer learned values to a new CNN, in this case anomaly detection for an 802.15.4 network.

The primary method used entails “freezing” the first layers of the pre-trained network and using these layers as a “feature extractor,” but plugging in and training entirely new, “unfrozen,” randomly initialized weight layers to replace the last layers of the network, which will perform classification based on those extracted features. This method assumes that the important features to be extracted from an input to perform an accurate classification of that input are the same for both the base and target tasks, but that the correct classification based on those extracted features may differ between the base and target tasks.

We implemented the following eight different variants:

1. A special placeholder value that denotes training from scratch (no TL, random weight initialization, all layers unfrozen)
2. Unfreeze only the last fully connected layer
3. Unfreeze all (3) fully connected layers
4. Unfreeze the later convolutional layers (the second one) and all fully connected layers
5. Unfreeze all (2) layers (convolutional and fully connected) in the entire network
6. Scenario 1, but reset the unfrozen layers
7. Scenario 2, but reset the unfrozen layers
8. Scenario 3, but reset the unfrozen layers

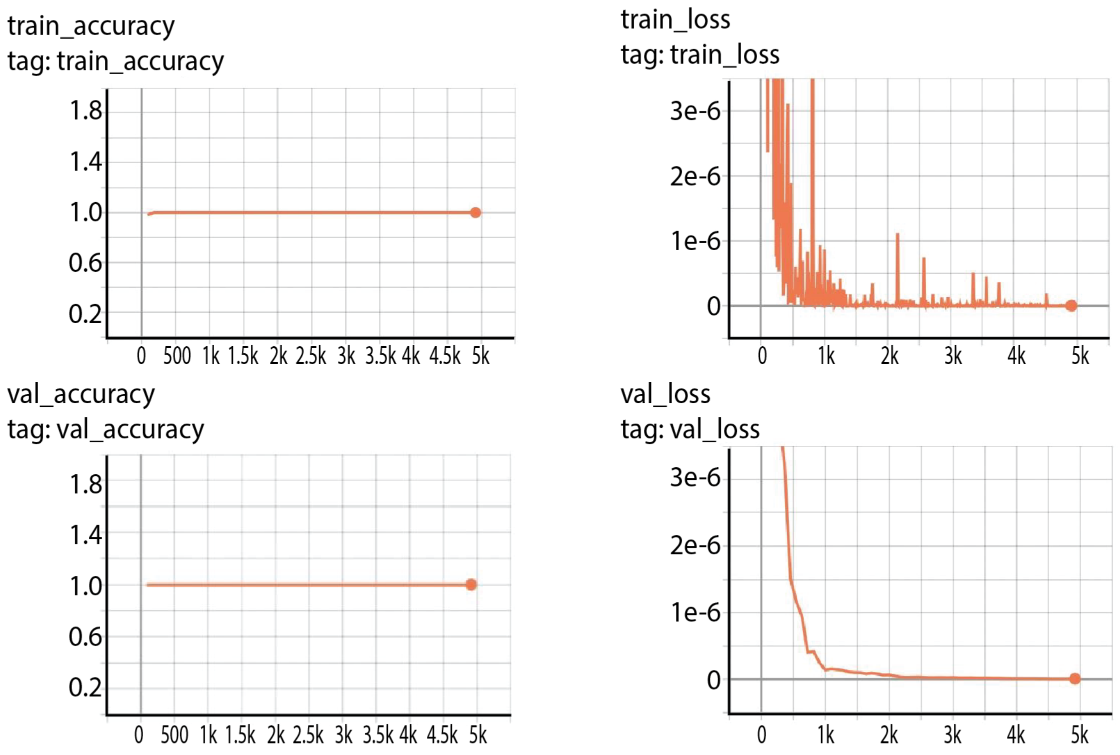


FIGURE 6. For these Bluetooth anomaly detection model results, the Braktooth anomaly [8] was used to create a flooding attack on a Bluetooth speaker. Graphs are depicted for the results of training accuracy, value accuracy, training loss, and value loss.

The WIMS software includes tunable per-layer learning rates so that unfrozen layers can learn at different rates depending upon whether their weights have been reinitialized prior to TL. For this application, we want larger learning rates for training from scratch and smaller learning rates for transfer. This allows the system to learn quickly from scratch and fine-tune existing knowledge carefully during transfer. The system also supports different L2 regularization penalties used by the adaptive moment estimation (ADAM) optimizer [7]. ADAM is a computationally efficient gradient-based optimization algorithm commonly used to fine-tune weights in neural networks. WIMS also uses different learning rate scheduling policies based on the layer type—“base” meaning unfrozen and reset (or learning from scratch in scenario 1), or “transfer” meaning unfrozen but not reset.

The system uses k-fold cross validation when training models to generate checkpoints, and when testing those models or using them together to vote and make a collective prediction. The k-fold cross validation randomly splits the training dataset into 10 equally sized subsets. Ten networks are then trained where each network uses a different 1/10 of the data for validation and the other 9/10 for training. When testing and making predictions, the 10 model checkpoints work together to vote and determine if a given sample is anomalous or non-anomalous. They each get an equal vote, and the average of their confidences that a given sample is anomalous is used to make a collective prediction, with a threshold of 0.5 (>0.5 means anomalous, <= 0.5 means non-anomalous). This was experimentally determined to be critical for achieving accurate results.

When generating models using TL, several questions arise when training models to detect a given anomaly: Which protocol should we start with (i.e., train from scratch on)? Which existing checkpoints should we use to train new models using TL? In which order should we train on protocols—can we use a previous protocol’s TL checkpoint as a base for further TL on a new protocol?

To address these challenges, we developed a tree-based algorithm for exploring the optimal scenario. Each tree has a single protocol at its root; we train from scratch on that protocol. The tree is traverse

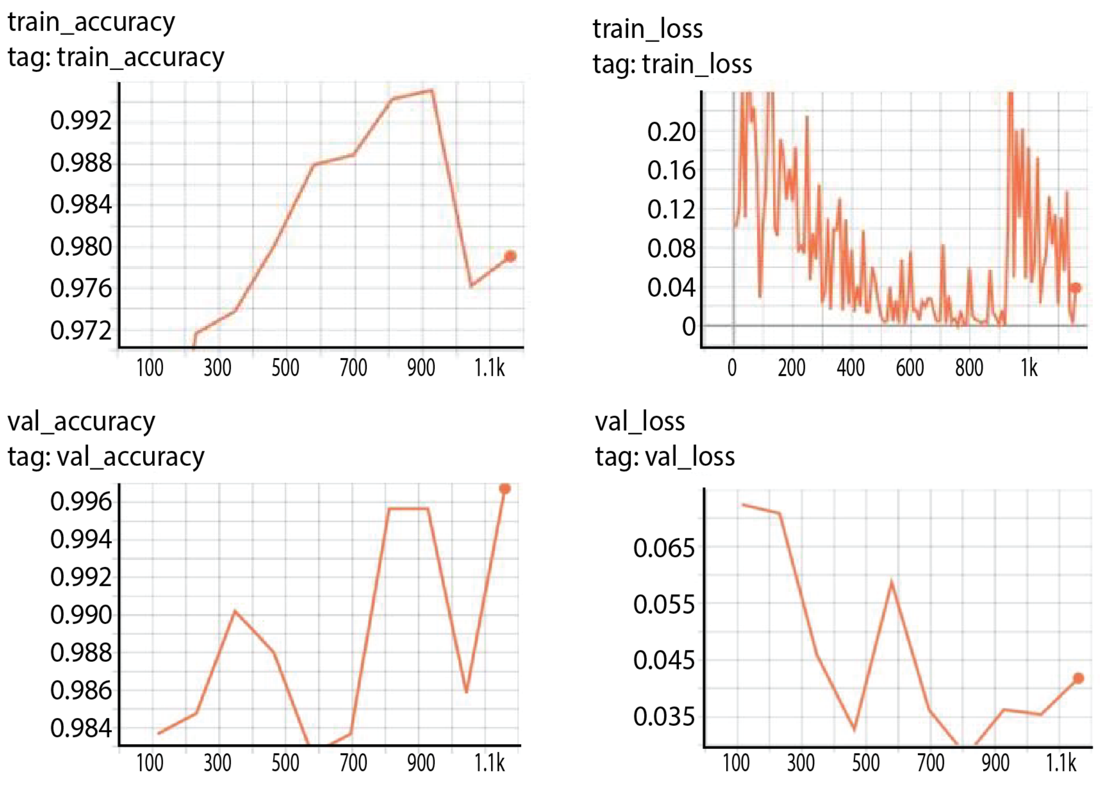


FIGURE 7. In our results, TL was used to apply Bluetooth flooding anomaly models to an 802.15.4 protocol. Graphs are depicted for the results of training accuracy, value accuracy, training loss, and value loss.

TL from a parent node to generate its child node(s). All methods of TL are compared, combinations of frozen, unfrozen, and "reset" layers. At that point, the algorithm selects the best F1 score to occupy a tree node going forward before moving to the node's children. Each tree contains all possible TL checkpoints starting from a base protocol without any cycles—an ancestor in the tree is never revisited as a child. Multiple trees can be produced to see which protocol makes for the best base.

Evaluation

We tested the above classification and prediction algorithms and then used the TL techniques described above and experimented under a variety of conditions. For the Bluetooth protocol, we used a Bluetooth speaker and the Braktooth anomaly [8]. This causes a response-flooding result—the speaker is disconnected from Bluetooth and then reboots. The three features selected for the stackable CNN were packet interarrival time, packet size, and packet type. The results are shown in figure 6. After fine tuning training parameters, we were able to achieve 1.0 accuracy. The results also show training and testing loss.

We then performed TL to the 802.15.4 protocol. We ran tests using a wireless personal area network (WPAN) consisting of an 802.15.4 hub and a thermostat, and we implemented a flooding response. Using combinations of the eight TL scenarios, we were interested in the two following figures of merit: 1) can we achieve high levels of accuracy, and 2) can we reduce the training time?

Figure 7 shows the results for one of these scenarios. As can be seen, after only nine training epochs, we obtained prediction accuracy of 0.97.

By observing training and target loss it is also possible to observe periods of model instability during gradient update epochs. To better understand this issue, we developed and implemented numerical function fitting software for IoT data for 12 different probability distributions, including log-gamma, parteo, and exponential. They revealed fundamental structural differences between Bluetooth and WPAN for the features we selected.

From an algorithmic perspective, the project achieved its objectives of demonstrating TL for multiple wireless protocols. Enabling factors that contributed to the success of the TL include the development of stackable GAF image production techniques, a deeper understanding of how to fine tune CNN parameters for prediction, the use of the eight different scenarios for TL, and the tree-based approach for TL protocol selection. Future work could explore different types of IoT-based attacks such as man-in-the-middle and the use of adversarial ML techniques to enhance our understanding of poorly understood protocols. We also expect that the probability distribution fitting tool will be extremely useful in understanding new datasets.

Conclusion

This paper described WIMS, a distributed system used to monitor deployed IoT networks at the enterprise level. It uses COTS IoT protocol analyzers/sniffers and SDRs to collect wireless network traffic. On a device-by-device basis, packet captures are turned into images. A CNN is trained to recognize normal versus anomalous images. Upon detection of an anomalous image, alerts are generated.

One of the notable aspects of WIMS is the use of TL. This approach reduces the time to train a CNN on new protocols and new attack vectors. Our current plan is to continue to classify new IoT protocols as they come online, as well as new types of IoT cyber threats.

References

[1] Link Labs. "The complete list of wireless IoT network protocols." 2016 Feb 8. Available at: https://www.link-labs.com/blog/complete-list-iot-network-protocols.

[2] 3GPP. "5G system overview." 2022 Aug 8. Available at: https://www.3gpp.org/technologies/5g-system-overview.

[3] Amazon.com Inc. "What is Amazon Sidewalk." *Developer Guide: AWS IoT Core.* [Accessed online 2023.] Available at: https://docs.aws.amazon.com/iot/latest/developerguide/amazon-sidewalk-overview.html.

[4] Spanalytics, LLC. "PANalyzer." [Accessed online 2023.] Available at: https://spanalytics.com/product/panalyzr/.

[5] Ellisys. "Ellisys Bluetooth Vanguard advanced all-in-one Bluetooth analysis system." [Accessed online 2023.] Available at: https://www.ellisys.com/products/bv1/.

[6] Teledyne Lecroy. "Frontline X500 wireless procotol analyzer." 2023. Available at: https://cdn.teledynelecroy.com/files/pdf/frontline-x500-datasheet.pdf.

[7] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." ArXiv preprint; arXiv:1412.6980 (2014). Available at: https://arxiv.org/abs/1412.6980.

[8] Garbelini ME, Chattopadhyay S, Bedi V, Sun S, Kurniawan E. "BRAKTOOTH: Causing Havoc on Bluetooth Link Manager." Asset Research Group. Available at: https://asset-group.github.io/disclosures/braktooth/.

Malware Bytes

James Holt, Edward Raff

Over the past decade, the Advanced Computing Systems (ACS) office of the Laboratory for Physical Sciences (LPS) has explored a wide variety of approaches to analyzing and classifying binary files. One of the most significant findings, and a consistent theme of our work, is the surprising effectiveness of algorithms that look directly at the bytes of files. Previously, most approaches to applying machine learning (ML) to binary files first extracted some set of features, and used these features as input to an ML model. In contrast, we have built ML models that learn directly from the file itself, ingesting and learning from the file as a sequence of bytes.

One advantage of this is that it requires no domain-specific or file-type-specific knowledge, so the exact same approach and set of tools can be used on multiple file types. Because of this, we have been able to use the same tools, with no alteration or customization, to build classifiers for Windows executables, Linux executables, PDFs, Microsoft Office files, rich text files (RTFs), and others.

[Photo credit: iStock.com/Quardia]

Historically, cybersecurity tasks concerning binary files, such as malware detection, analysis, triage, reverse engineering, disassembly, decompilation, etc., have required deep expertise and a low-level understanding of the structure of files and function of operating systems. The tools that have evolved to perform or assist with these tasks also require deep expertise to create, to use, and to keep up to date. The complexity of these files and systems, and therefore the degree of expertise needed to work with them, continues to grow, and this creates challenges in training people, in keeping the software tools current, and in computational cost of running the tools.

This challenge is not going away. Tools which embody deep knowledge and understanding are certainly still necessary. But, we can now complement them with no-domain-knowledge tools that can be rapidly adapted to new problems, providing agility, speed, and scale. In some cases they can reduce the need for more expensive high-domain-knowledge tools. Along these lines, ACS has developed a portfolio of byte-based algorithms, including compression-based file similarity metrics, a highly efficient n-gram algorithm, convolutional neural network-based file classification, n-gram and logistic regression (LR)-based file classification, and automatic signature generation.

In this article we will share some highs, lows, and points of interest from the journey our research has taken over the last decade, exploring different forms of ML for file classification, testing existing ML techniques, innovating new ones, creating new algorithms that yielded 100-fold performance gains, curating datasets, and delivering new capabilities. Along the way, we published many of our advances in academic papers. Often these were accompanied by code, allowing others to leverage and benefit from our new algorithms. As we go through these efforts, we will highlight the published papers for the reader who would like to go deeper.

The false negative/false positive trade-off

Antivirus (AV) products have been deployed and in use for decades. They were originally targeted at home consumers, which imposed early design constraints. The AV product needed to run fast enough to not get in the way (too much). If the AV slowed the system down so that it was no longer interactive, making users wait too long to load or use applications, they would forgo the product entirely. Essentially, there is a friction ceiling that AVs need to avoid to keep users happy.

This friction ceiling applies for both speed and accuracy. Imagine the AV was more likely to predict a benign file as malicious (i.e., false-positive or FP) than a malicious file as benign (i.e., false-negative or FN). This would mean every new benign application would have a larger chance of causing an alert to the user. This would result in interruption of their work, remediation, and whitelisting—that is, friction. Again, the user would forgo the AV. To prevent this, AVs are designed around minimizing false positives. These are reasonable design constraints, motivated by real-life use, and still relevant today. For this reason, these ideas and goals are heavily embedded in the current industry and academic cultures.

Unfortunately, systems designed this way do not adequately address the breadth of issues we see in the government. Some missions require that every file be inspected, analyzed, and a determination of risk made. In such a case, the FP/FN trade-off is not relevant, and the coarse "yes/no" returned by many AVs does not allow us to adjust this trade-off as we need [1]. Instead, what is desired is a score that can be used to rank all the files from most-likely malicious to least-likely. You can quantify this using the area under the curve (AUC), which measures the quality of a "ranking" of files by maliciousness. In other cases, like forensic investigations, there is an informed presumption that malware already exists and needs to be found. In this case, the motivations and costs are different. Minimal FN is the goal, so that if malware is actually on the system, it will be found, not missed, and evidence gathered. The costs of FPs are manageable in this instance, so the incentives are reversed. Our research has explored approaches for missions with varying FN/FP sensitivities.

Dealing with the fire hose of files

AV software on an individual user's machine is one piece of a larger puzzle. Large AV vendors have millions of installations of their software, all sending suspicious files back to a central system for analysis. It may be useful to think of the user's AV as the front end, or the collector, and the central system as the back end. This central system will receive millions of files daily. From the view of the central system, this

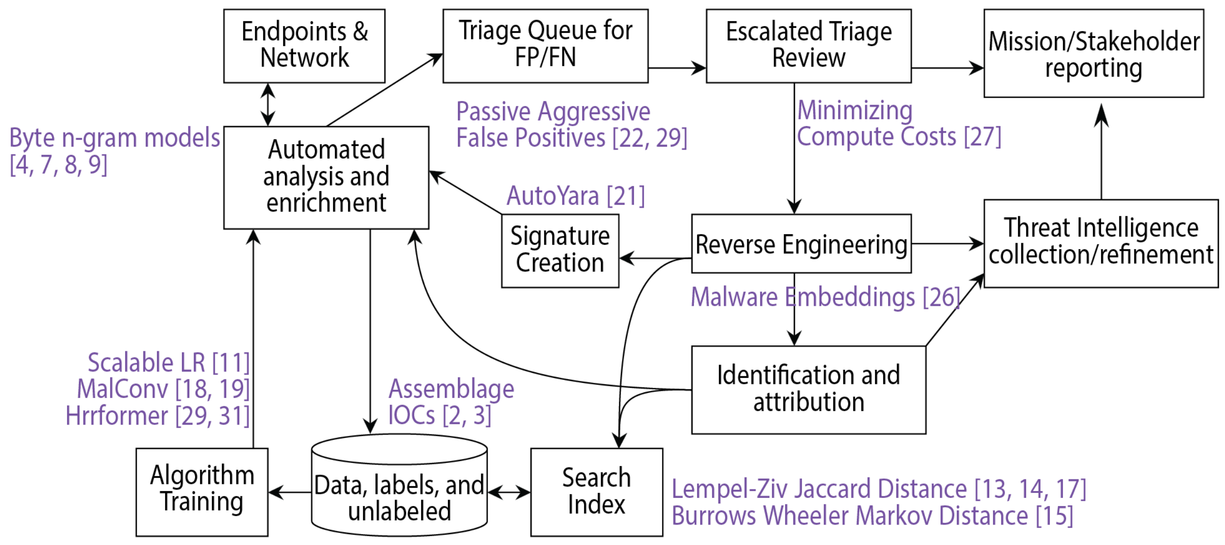

Endpoints & Network
Triage Queue for FP/FN
Escalated Triage Review
Mission/Stakeholder reporting
Byte n-gram models [4, 7, 8, 9]
Automated analysis and enrichment
Passive Aggressive False Positives [22, 29]
Minimizing Compute Costs [27]
AutoYara [21]
Signature Creation
Reverse Engineering
Threat Intelligence collection/refinement
Malware Embeddings [26]
Identification and attribution
Scalable LR [11]
MalConv [18, 19]
Hrrformer [29, 31]
Assemblage
IOCs [2, 3]
Algorithm Training
Data, labels, and unlabeled
Search Index
Lempel-Ziv Jaccard Distance [13, 14, 17]
Burrows Wheeler Markov Distance [15]

FIGURE 1. A simplified diagram of the kinds of interactions and stages necessary in supporting large scale file analysis. The purple text highlights selected research results, placed next to the stage it impacts.

fire hose of files must be triaged. Decisions must be made about whether and how to analyze each file, which analysis tools will provide the most relevant information, and how much computing and human effort to spend on each file. The US government faces similar issues, as it operates and defends millions of computers.

This large-scale analysis process can be thought of as a pipeline, including a variety of feature extraction and analytic processes which feed into decision-making. Often, analytics with increasing cost or levels of effort are applied to subsets of the files as they progress through the pipeline. Building capabilities that enhance both the analysis and decision-making parts of this pipeline has been a goal of our research.

Figure 1 is a simplified representation of parts of the file/malware analysis pipeline, showing how files or other information flows between stages. The purple text indicates papers we have published that are applicable to various stages.

Helping our academic and industry partners understand the government's perspective on this pipeline, and analytic needs associated with it, has also been a goal. We hope an increased understanding of this will motivate and incentivize additional research, resulting in new and improved capabilities for all cyber-defenders.

Deep file analysis

You've discovered some malware, what happens next? Sometimes you can just delete it, clean the machine it was on, and be done. Other times you may want to create a signature to automatically detect it, if it shows up again. If the malware was found on a high-value asset, you may want to investigate more deeply. How did this get onto this machine? What is the goal of this malware? How does it work? Who made it? How does the owner of the malware communicate with and control it?

The computers of the US government and defense contractors are targeted every day by actors from around the world. Some of these actors are highly motivated and well-resourced to come after our information. At times they even create custom-crafted malware that is specific and goal oriented. It is often critical to understand how we are being attacked, by whom, and what their motivations are. Better understanding of these factors informs our steps to remediate intrusions and prevent future ones through signatures and security updates.

Performing custom deep analysis on a specific file is a labor-intensive and primarily human-driven process. This work can take days or weeks for a single file, consuming analysts' time and creating a backlog. This slowness also interferes with a critical function of the analyst: noticing trends and making connections. Is this malware similar to something we've seen before, or related to something we've seen before? That information could help reverse engineering go faster, avoid redundant work, and inform threat intelligence [2, 3]. Instead of treating all individual malware occurrences as isolated, understanding recurrence of related malware could help us identify a prolonged campaign.

Our foray into malware detection

Starting in 2012, the ACS team began looking at malware detection as a way to apply our ML research to real-world problems. At the time this seemed eminently reasonable. After a review of the academic literature, it seemed many published papers reported over 99 percent accuracies using byte n-grams. Given the reported results, it seemed like the easiest way to get started. Other approaches like dynamic analysis, disassembly, and similar techniques required extensive infrastructure, making both training and deployment more challenging. The reported accuracies in the literature were also not meaningfully different, so why pay this high price?

Replicating the data collection, algorithm training, and design processes allowed us to produce initial results. Significant software engineering work was done to make the process as fast as possible, rewriting research-grade Python code into Java gave us 100-1000 times speedups, and made it easy to deploy in a variety of mission environments. We had great initial success in our lab environment, so we decided it was time for field testing with our mission partners. As our first trial deployment reached its conclusion, we got harsh feedback: *the system was not useful, everything was labeled as malware, and they did not want to continue using the tool.*

This is how we first learned of the considerable gap between academic knowledge on the subject of malware detection and the practical application of such knowledge. The crux of the issue was that all our benign data came from a clean installation of Microsoft Windows, and our model would learn to literally look for the bytes of "Copyright Microsoft Corporation" as its method of determining "malicious" versus "benign." In reality, we had developed a "from Microsoft or not" detector. This underscores the importance of knowing your data, and training with representative data, as an ML model can only learn from the data you give it.

It was clear that we needed to get better-quality benign data for our research efforts. It was also clear that nontrivial parts of academic literature and culture around the application of ML for malware had misunderstandings of what would generalize to practice, and what tools and volumes of data were needed.

Byte-based models

Given the cost of feature engineering and domain knowledge, our team decided to explore just how far byte-based models could be pushed. This led to three different families of techniques that we have advanced over the past decade: 1) byte n-gram-based models, which have been surprisingly effective for malware detection; 2) using compression algorithms (similar to those used in a .zip file) for ML, which we have found especially useful for information retrieval situations; and 3) scaling and accelerating deep-learning techniques to handle malware data.

All three techniques have the broad benefit that we can apply them to any file format or data type, provided we are able to obtain sufficiently representative data (both benign and malicious) for said file format. This has allowed our small team to adapt and stay relevant to mission interests despite gaps in our domain expertise for many platforms that our mission partners are interested in. Each is also useful from a deployment and product life cycle perspective: because we work from raw bytes, there is no parsing involved. This means we never have parsing errors in production, a common issue as malware authors intentionally subvert the fact that the specification doesn't always match the actual implementation for a given file loader. This also avoids highly cumbersome dependencies. Many projects depend on the parsing capabilities of LIEF, IDA Pro, Ghidra, or other software with significant domain-expert knowledge. While valuable, that also means these tools are constantly evolving as malware actors adapt to detection approaches. This continuous evolution results in a significant technical burden of keeping tools current.

Byte n-grams

When we started working on byte n-grams with higher-quality data, one early win became very important for people using our algorithms and code. Deployment was fast. The model size and the code to run it was only a few megabytes. The model essentially contained a list of byte 6-grams, and corresponding weights indicating if the byte pattern was positively/negatively impacting the malware decision [4]. This was low profile, meaning it could be deployed on almost any platform. Running the model could be done using a small buffer, streaming the data into main memory and checking if bytes occurred, incrementing/decrementing the cumulative score accordingly, and then returning the answer. Using rolling hash functions means you could perform the initial load of a file into memory or transfer it to another location and get a benign/malicious prediction essentially "for free" because this code was faster than the data transfer process.

Our first deployments were in situations where speed was the critical factor to deployment and mission success. Anything that took more than 100 milliseconds would be too slow to run on all the incoming data. Others were mission partners that needed to go on "fly away" missions, carrying all the computing they could use with them, and our tool was fast and easy to add. More deployments came as some missions used an ensemble of products to make a prediction, and our byte-based model was unusual compared to the standard AV approaches. This is valuable, as ensembles improve their predictive performance when the members of the ensemble are different (or, uncorrelated in their errors).

These benefits do not come for free though. The first cost is a decrease in accuracy compared to many industry products, which was expected. Industry AV products are built using vast amounts of data, and many person-years of labor analyzing and signaturing specific malware. AVs, by virtue of their presence on the host system, also have the ability to observe more than just the file, including behavioral patterns. Our byte-based models do not have these advantages, but they address different strategic needs than AVs.

The real costs come in algorithmic issues. Creating byte n-grams is expensive. We were using a cluster of 12 machines over three weeks to perform the n-gram mining on around 400,000 files[a]. Our method of choice, after considerable testing, was L_1 penalized logistic regression, because it could perform feature selection as a part of model training (the fewer features we have, the smaller the model for deployment, and the faster it is).

This led to years of research problems to tackle. First was discovering why byte n-grams failed to perform as well as we expected, and noting that the "Copyright Microsoft" problem had gone unchallenged in academic literature for decades [4]. We learned that byte n-grams are not 99.9 percent accurate; but surprisingly, they perform better than disassembly-based n-grams [5] and have more robustness to alterations than many AV products [6]. This increased our confidence in them. Addressing the large computing resources required to generate n-grams for a dataset, we developed new algorithms, including our KiloGrams algorithm, which reduced the $O(256^n)$ complexity down to just $O(n)$ [7, 8, 9]. This allowed us to explore larger byte n-grams. Our current constraint to additional scaling is the memory required to train logistic regression models on multi-terabyte datasets. We continue research on how to scale these computations [10, 11].

Compression-based similarities

A second direction we have explored is using compression algorithms to measure the similarity between arbitrary byte sequences. This is inspired by Kolmogorov complexity and originally developed as the normalized compression distance (NCD) [12]. To understand this, consider the function $K(\cdot)$, which takes an input x, which has a length of $|x|$, and returns the length of the smallest program that can reproduce the value of x. So, imagine you have the input x = *aaaaaaaaaaa*. Then K(x) would return a program that loops 11 times, returning the value of "a" each time. This function is thus the perfect compressor. The ratio $|K(x)|/|x|$ tells you how compressible a sequence x is. You can extend this idea by giving the function inputs, so $K(x|y)$ would be the shortest program that produces x as output, given y as input. There is always the option to ignore the input, so we get $K(x|y) \leq K(x)$.

Computing a perfect K(x) is impossible, but we have general-purpose compression algorithms like zip, bzip, lzma, etc., which approximate K. If we represent our favorite compression algorithm, C(x), which computes the size of the string x after being compressed, we can compute a normalized compression distance (NCD):

$$NCD\,(x, y) = \frac{C\,(x + y) - \min\{\,C\,(x), C\,(y)\}}{\max\{\,C\,(x), C\,(y)\}}$$

This gives us a way to measure the similarity between arbitrary byte sequences. This is useful for

a. This was also after years of extensive performance optimization of our n-gramming software.

malware because very few ML techniques can scale to the size of an executable. For example, a 200 megabyte (MB) program is not unreasonable, but this is larger than the entire CIFAR100 dataset. But we can and do run compression algorithms without issue on large files, and this trick requires no training data. It is also useful for forensic applications: it may be the case that a recovered hard drive has corrupted data, and so only fragments of the file exist. Was the drive corrupted because of malware or just hardware failure? If you can match a fragment of a file to known destructive malware, you can answer the question. Similarly, an in-progress download may be cut short by other defenses, but you want to see if the fragment of the download matches any known data. Using compression similarity, we can still do these tasks, even though domain-knowledge-based parsing becomes impossible.

Unfortunately, NCD does not scale up. You must run a compression search for every distance computation you want to perform, which is painfully slow. However, this inspired us to try to optimize the idea behind NCD: repurposing compression algorithms for ML.

This started by noting that Lempel-Ziv-based compression algorithms, which build compression dictionaries, tend to work best for malware detection. We took the compression dictionary created by the compression process, and treated it as a set of objects representing the file. Then we used set similarity measures like the Jaccard distance to measure the distance between two files. This way we could build the compression dictionary once per file, and re-use it in distance computations. Combined with min-hashing to make Jaccard distances faster, and with provably bounded error, the Lempel-Ziv Jaccard distance (LZJD) was born [13]. LZJD can compare any two sequences of bytes (e.g., files) and produce a similarity score. LZJD led to early success in mission deployments as it allowed the same kind of functionality as hashing digests such as ssdeep and sdhash, while being many times faster to search and more accurate [14]. With an eye to scaling up to even larger datasets, we applied the same trick using the bzip compression technique. Instead of Lempel-Ziv, bzip uses the Burrows-Wheeler transform and run-length encoding, which we connected to being spiritually similar to a Markov model: you assume that the current token is sufficient to predict the next token. This led to the Burrows-Wheeler Markov distance (BWMD). It functions similarly to LZJD, and we were able to scale to searching millions of files in seconds [15]. We also extended LZJD to create feature vectors usable for ML and added the ability to over-sample the minority classes to help handle class imbalance that often occurs in family detection work [16, 17].

Deep learning over long sequences

As mentioned, a 200 MB executable is not unusual. But processing that byte-by-byte means a 200,000,000 step long time series classification task. Most recurrent neural networks and long short-term memories consider 5,000 steps to be “very long,” and only recently transformers have attempted to get to around 32,000 steps—at a nontrivial cost to accuracy.

When we first attempted to tackle this problem back in 2016, we turned to convolutional networks as an approach to tackling the scale of a single data point. Through significant exploration, we found that the canonical best practices did not work for malware data. We had to make our model shallower, with wider convolutions, and eschew normalization layers to get improved accuracy. Our first result, MalConv, could process 1,000,000 steps, a significant advancement over the previous best possible number[18]. But that still wasn’t enough. A few years later, we cracked the 200,000,000 time step barrier for effectively length-invariant convolutional neural-network-based training [19].

While there have been barriers to deploying the MalConv architecture operationally, it has been a significant advancement that other organizations have built on. Academically, we have seen uses of MalConv for authorship identification, tool-chain identification, and other reverse engineering tasks, with similar deployments in the national labs. MalConv has also been one of our most successful efforts in influencing academia to focus more effort on problems that will have a real-world impact. Notably, a significant amount of work in adversarial attacks and defense possibilities has been developed using MalConv, providing us with valuable information about potential weaknesses, and has allowed us to develop new non-negative techniques to mitigate some of these attacks [20]. The research community’s work exploring adversarial ML attacks on MalConv has provided insights into vulnerabilities and potential defenses of a class of ML methods which would have taken us years to work out on our own. This symbiotic information benefit is one of the major reasons we push to continue working publicly with academic partners.

Hybrid full-stack

The "domain-knowledge free" approach our team started has invigorated significant academic research that helps us better understand what capabilities and risks are possible in the near future, while producing valuable, fast, low technical debt solutions to satisfy a variety of Agency missions. But this approach is perhaps most valuable as the first-level analysis/automated triage. Eventually, more technically sophisticated resources are required. Our goal over the past few years has been to leverage what we've learned and begin to adapt it toward hybrid solutions, forward integrating it into the set of common tasks that take up valuable analyst time.

Many of the techniques we have developed have been deployed in the government directly or via corporate partners who have leveraged the research. They address pieces of the big picture in figure 1. Another application of our extremely efficient n-gramming techniques is *AutoYara,* a technique for taking a handful of files known to be related and automatically producing a Yara signature that can be run with standard tools. Compared to human analysts doing this work, we found AutoYara could produce satisfying signatures for 84 percent of their work queue. This is especially helpful when human analysts have items flowing into their work queue faster than they can work them. Top human analysts produce better signatures, but AutoYara can be scaled to handle the volume that humans cannot [21].

Cyber environments vary widely. For ML models that are used in production, one may need to adapt the model to the individual environment that they are operating in—because models trained on a global population of benign and malicious programs often don't quite fit the idiosyncrasies of a specific environment. We developed a *passive aggressive* approach to patching a deployed malware detector to allow it to be adapted or tuned to a local environment with limited data [22].

Working with collections of files that may number in the hundreds of millions creates many storage and computational challenges. For any single task that is known a priori, one can extract the necessary features from each file and then discard the file. But we don't always know what information in a file will be important to future analyses. Therefore, creating a compact representation of a file, which contains information that can be used for multiple tasks, such as malware detection, family classification, and malware attribute prediction, is desirable. In ML lingo, can we use metric learning to derive low-dimensional embeddings that are useful for downstream tasks? We explored this direction, learning embeddings using CAPA [23, 24] and Endgame Malware Benchmark for Research (EMBER) [25] features, and experimented with different kinds of loss functions. This direction has potential, but more work remains to be done [26]. Similarly, we've used ML to predict what CAPA will find, with the ability to abstain from predictions when uncertain. This allows us to *minimize compute* costs by running expensive analytics only when necessary and can save multiple compute years of time if expensive virtual machines are used in the second phase [27].

Cognitive science-inspired learning

Cognitive science, the general study of how the brain and human cognition work, played a fundamental and important role in early artificial intelligence research. Current work in deep learning has its origins in neural networks inspired by the brain, though it has diverged much since that original spark. One of our research directions has been to look at tools that have found utility in cognitive science and try to move them back into ML. The hope is that this could allow us to obtain benefits, while being different from existing approaches, and enhance the diversity of our models.

The holographic reduced representation (HRR) is one particular method our team has extended. Originally developed in 1995 [28], it found significant use in cognitive science for its ability to perform symbolic artificial intelligence on a biologically plausible substrate. If $F(\cdot)$ represents the Fourier transform, and $F^{-1}(\cdot)$ its inverse, and *a, b, x, y* are all vectors in a *d*-dimensional space, it was shown that you can develop a binding operation $a \otimes b = F^{-1}(F(a) \odot F(b))$. This has unique properties where you can distinguish between items added and bound together using an inversion operator $\dagger$, so a "statement" $S = a \otimes b + x \otimes y$ can be constructed, and you can approximately determine "what was bound with *x*" by computing $y \approx S \otimes x^{\dagger}$. It turns out that this is sufficient to perform symbolic artificial intelligence, but all the operations are being done with real-valued vectors and in a compressed representation.

The HRR presented a new direction to build more efficient algorithms with potentially new capabilities. We decided to explore this direction, and we have had some success. First was the task of determining how to modernize HRRs within a differentiable framework so that we could use modern PyTorch and JAX libraries [29]. This enabled multiple new directions. First, we leveraged the symbolic properties of HRRs to build a neural network that performed most of its work on a third-party, and slightly untrusted, computer. This allowed for reducing the amount of local compute resources while hiding the nature of the data from the third party via a kind of pseudo-encryption. While not as strong as encryption, these *Connectionist Symbolic Pseudo Secrets (CSPS)* were several thousand times faster than the existing cryptographic options [30]. More recently we used HRRs to tackle the computational bottleneck of transformers. Our malware data has sequences in excess of T = 200,000,000 time steps, and normal transformers have $O(T^2)$ computational complexity. Using HRRs, we brought this down to just linear $O(T)$ cost by inventing the HRRformer, while simultaneously improving accuracy [31].

In both cases, we were unable to fully solve the problem of interest, which is often the case in research; however, both cases significantly informed our computational needs for the future. The CSPS informs the possibility of using the floating point capabilities to perform at least some limited set of obfuscation when full cryptography may not be required or feasible. The HRRformer brought us closer to the computational throughput needed to tackle our long sequence malware goals—now we need an additional two orders of magnitude improvement, where before we needed seven. Our research thus transformed an unlikely avenue for computing systems to impact future missions into something with realistic potential in the next few years.

Which ML techniques really work?

When considering future hardware designs, the time from conception to completion is measured in years. For this reason, it is critical that we design algorithmic primitives we know will be useful several years into the future, and discover which do not scale to our desired needs [32]. Working in malware detection research helps us to ensure that is the case, by narrowing the focus to ML techniques that generalize.

This is an insight we have developed over time through hard-fought progress. Most ML and deep-learning research occurs in computer vision, natural language processing, and speech/signal processing. In all of these domains, there is a shared fundamental property: things near each other are related to each other. This is the prior of spatial locality, which is a core foundation of convolutions, recurrent neural networks, and positional embeddings. When this prior is too strongly violated, many of the “best” techniques that the community “knows” work begin to fall apart. In pushing these boundaries, we separate what is only conditionally useful from what truly generalizes to new and different problems.

For example, we recently turned to multiple instance learning (MIL) as a technique that makes the ML match how analysts do their job: something is benign by default, and only becomes malicious if something malicious is detected. There are no “positive” features in this regard, or at least, there shouldn’t be. Yet we found that many current deep MIL algorithms fail to maintain this invariant and cause us unexpected and unsatisfying results [33]. Similarly, we must be careful with our data, and so have spent time studying how to evaluate [34] and construct [35] new and more informative datasets so that we, and the community at large, confidently know what we are measuring.

This process has helped us further refine the kinds of fundamental deep-learning building blocks that we care about for designing future hardware. It informed our big-picture view on what algorithmic issues we can work around and what are still roadblocks, compute versus memory trade-offs, and the types of primitives and operations we find consistently useful in malware and the broader ML problem spaces.

Conclusion

Our team has made significant advances in scaling, accelerating, and parallelizing ML techniques which can be applied to malware detection and a host of other domains and problems. We function as an interface between academic research, industry, and government mission users, following the latest research, influencing the directions of academic research and commercial investments in research and development, advancing the state of the art with our own work, and bringing the best of all of these to bear on critical mission problems. And we look forward to doing this for years to come.

References

[1] Nguyen AT, Raff E, Nicholas C, Holt J. "Leveraging uncertainty for improved static malware detection under extreme false positive constraints." In: *IJCAI-21 1st International Workshop on Adaptive Cyber Defense;* 2021.

[2] Pingle A, Piplai A, Mittal S, Joshi A, Holt J, Zak R. "Relext: Relation extraction using deep learning approaches for cybersecurity knowledge graph improvement." In: *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining,* New York, NY; 2020: pp. 879–886. Association for Computing Machinery. Available at: https://doi.org/10.1145/3341161.3343519.

[3] Piplai A, Mittal S, Joshi A, Finin T, Holt J, Zak R. "Creating cybersecurity knowledge graphs from malware after action reports." *IEEE Access.* 2020;8:211691–211703. doi: 10.1109/ACCESS.2020.3039234.

[4] Raff E, Zak R, Cox R, Sylvester J, Yacci P, Ward R, Tracy A, McLean M, Nicholas C. "An investigation of byte N-gram features for malware classification." *Journal of Computer Virology and Hacking Techniques.* 2018;14:1–20. Available at: https://doi.org/10.1007/s11416-016-0283-1.

[5] Zak R, Raff E, Nicholas C. "What can N-grams learn for malware detection?" In: *2017 12th International Conference on Malicious and Unwanted Software;* 2017: pp. 109–118. Available at: https://doi.org/10.1109/MALWARE.2017.8323963.

[6] Fleshman W, Raff E, Zak R, McLean M, Nicholas C. "Static malware detection & subterfuge: Quantifying the robustness of machine learning and current anti-virus." In: *2018 13th International Conference on Malicious and Unwanted Software;* 2018: pp. 1–10. Available at: https://doi.ieeecomputersociety.org/10.1109/MALWARE.2018.8659360.

[7] Raff E, Nicholas C. "Hash-grams: Faster N-gram features for classification and malware detection." In: *Proceedings of the ACM Symposium on Document Engineering 2018,* New York, NY; 2018. Association for Computing Machinery. Available at: https://doi.org/10.1145/3209280.3229085.

[8] Raff E, McLean M. "Hash-grams on many-cores and skewed distributions." In: *2018 IEEE International Conference on Big Data;* 2018: pp. 158–165. Available at: https://doi.org/10.1109/BigData.2018.8622043.

[9] Raff E, Fleming W, Zak R, Anderson H, Finlayson B, Nicholas C, Mclean M. "Kilograms: Very large N-grams for malware classification." *Learning and Mining for Cybersecurity;* 2019. Available at: https://eda.rg.cispa.io/events/lemincs19/papers/paper_raff_etal.pdf.

[10] Lu F, Raff E, Holt J. "A coreset learning reality check." In: *Proceedings of the AAAI Conference on Artificial Intelligence,* 37; 2023. doi: 10.1609/aaai.v37i7.26074.

[11] Raff E, Sylvester J. "Linear models with many cores and CPUs: A stochastic atomic update scheme." In: *2018 IEEE International Conference on Big Data;* 2018: pp. 65–73. Available at: https://doi.org/10.1109/BigData.2018.8622172.

[12] Li M, Chen X, Li X, Ma B, Vitanyi PMB. "The similarity metric." *IEEE Transactions on Information Theory.* 2004;50(12):3250–3264. Available at: https://doi.org/10.1109/TIT.2004.838101.

[13] Raff E, Nicholas C. "An alternative to NCD for large sequences, Lempel-Ziv Jaccard Distance." In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* New York, NY; 2017: pp. 1007–1015. Association for Computing Machinery. Available at: https://doi.org/10.1145/3097983.3098111.

[14] Raff E, Nicholas C. "Lempel-Ziv Jaccard Distance, an effective alternative to ssdeep and sdhash." *Digital Investigation.* 2018;24:34–49. Available at: https://doi.org/10.1016/j.diin.2017.12.004.

[15] Raff E, Nicholas C, McLean M. "A new Burrows Wheeler transform Markov Distance." In: *Proceedings of the AAAI Conference on Artificial Intelligence 34(04);* 2020: pp. 5444–5453. Available at: https://doi.org/10.1609/aaai.v34i04.5994.

[16] Raff B, Nicholas C. "Malware classification and class imbalance via stochastic hashed LZJD." In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security,* New York, NY; 2017: pp. 111–120. Association for Computing Machinery. Available at: https://doi.org/10.1145/3128572.3140446.

[17] Raff E, Aurelio J, Nicholas C. "PyLZJD: An easy to use tool for machine learning." In: Calloway C, Lippa D, Niederhut D, Shupe D, editors. *Proceedings of the 18th Python in Science Conference;* 2019: pp. 101–106. Available at: http://dx.doi.org/10.25080/Majora-7ddc1dd1-00e.

[18] Raff E, Barker J, Sylvester J, Brandon R, Catanzaro B, Nicholas C. "Malware detection by eating a whole EXE." *AAAI Workshop on Artificial Intelligence for Cyber Security;* 2017. Available at: https://cdn.aaai.org/ocs/ws/ws0432/16422-75958-1-PB.pdf.

[19] Raff E, Fleshman W, Zak R, Anderson HS, Filar B, McLean M. "Classifying sequences of extreme length with constant memory applied to malware detection." In: *Proceedings of the AAAI Conference on Artificial Intelligence 35(11);* 2021:9386–9394. Available at: https://doi.org/10.1609/aaai.v35i11.17131.

[20] Fleshman W, Raff E, Sylvester J, Forsyth S, McLean M. "Non-negative networks against adversarial attacks." In: *The AAAI-19 Workshop on Artificial Intelligence for Cyber Security;* 2018.

[21] Raff E, Zak R, Munoz GL, Fleming W, Anderson HS, Filar B, Nicholas C, Holt J. "Automatic Yara rule generation using biclustering." In: *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security,* New York, NY; 2020: pp. 71–82. Association for Computing Machinery. Available at: https://doi.org/10.1145/3411508.3421372.

[22] Raff E, Filar B, Holt J. "Getting passive aggressive about false positives: Patching deployed malware detectors." In: *2020 International Conference on Data Mining Workshops;* 2020: pp. 506–515. doi: 10.1109/ICDMW51313.2020.00074.

[23] Mandiant. CAPA. Available at: https://github.com/mandiant/capa/.

[24] Ballenthin W, Raabe M. "capa: Automatically Identify Malware Capabilities." 2020. Mandiant. Available at: https://www.mandiant.com/resources/blog/capa-automatically-identify-malware-capabilities.

[25] Anderson HS, Roth P. "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models." 2018; ArXiv. Available at: https://arxiv.org/abs/1804.04637.

[26] Rudd EM, Krisiloff D, Coull S, Olszewski D, Raff E, Holt J. "Efficient malware analysis using metric embeddings." *Digital Threats: Research and Practice.* Accepted August 2023. Available at: https://doi.org/10.1145/3615669.

[27] Nguyen AT, Zak R, Richards LE, Fuchs M, Lu F, Brandon R, Munoz GL, Raff E, Nicholas C, Holt J. "Minimizing compute costs: When should we run more expensive malware analysis?" In: *Proceedings of the Conference on Applied Machine Learning in Information Security;* 2022. Available at: https://www.camlis.org/andre-nguyen-2022.

[28] Plate TA. "Holographic reduced representations." *IEEE Transactions on Neural Networks.* 1995;6(3):623–641. doi: 10.1109/72.377968.

[29] Ganesan A, Gao H, Gandhi S, Raff E, Oates T, Holt J, McLean M. "Learning with holographic reduced representations." In: *Advances in Neural Information Processing Systems;* 2021. Available at: https://proceedings.neurips.cc/paper_files/paper/2021/file/d71dd235287466052f1630f31bde7932-Paper.pdf.

[30] Alam MM, Raff E, Oates T, Holt J. "Deploying convolutional networks on untrusted platforms using 2D holographic reduced representations." In: Chaudhuri K, Jegelka S, Song L, Szepesvari C, Niu G, Sabato S, editors. *Proceedings of the 39th International Conference on Machine Learning, vol. 162 of Proceedings of Machine Learning Research;* 2022: pp. 367–393. Available at: https://proceedings.mlr.press/v162/alam22a.html.

[31] Alam MM, Raff E, Biderman S, Oates T, Holt J. "Recasting self-attention with holographic reduced representations." In: Krause A, Brunskill E, Cho K, Engelhardt B, Sabato S, Scarlett J, editors. *Proceedings of the 40th International Conference on Machine Learning, vol. 202 of Proceedings of Machine Learning Research;* 2023: pp. 490–507. Available at: https://dl.acm.org/doi/10.5555/3618408.3618431.

[32] Raff E, McLean M, Holt J. "An easy rejection sampling baseline via gradient refined proposals." In: *26th European Conference on Artificial Intelligence,* 2023. Available at: https://ebooks.iospress.nl/pdf/doi/10.3233/FAIA230483.

[33] Raff E, Holt J. "Reproducibility in Multiple Instance Learning: A case For algorithmic unit tests." In: *Advances in Neural Information Processing Systems;* 2023. Available at: https://proceedings.neurips.cc/paper_files/paper/2023/hash/2bab8865fa4511e445767e3750b2b5ac-Abstract-Conference.html.

[34] Patel T, Lu F, Raff E, Nicholas C, Matuszek C, Holt J. "Small effect sizes in malware detection? Make harder train/test splits!?" In: *Proceedings of the Conference on Applied Machine Learning in Information Security;* 2023.

[35] Joyce RJ, Raff E, Nicholas C, Holt J. "MalDICT: Benchmark datasets on malware behaviors, platforms, exploitation, and packers." In: *Proceedings of the Conference on Applied Machine Learning in Information Security;* 2023.

YELLOWTIGER: Exploring the Future of Binary File Analysis

Vulnerability, Exploitation, and Mitigation Team,
Laboratory for Telecommunication Sciences

YELLOWTIGER (YETI) is a tool built to aid in understanding executable files. Focused primarily on finding useful relationships between binaries, the data that YETI generates has applications across a variety of domains, such as identifying and understanding the purpose of malicious files, enhancing vulnerability analysis, and predicting authorship or provenance of the file.

One key area that YETI gives emphasis to is expanding the types of features that can be extracted from binary files. Strings, sequences of bytes, and fields extracted from the headers present in binary files are widely used because they are quick to access and easy to compare; YETI adds to these possibilities by allowing extraction of features that require deeper understanding of the binary, such as the structure of the code. An understanding of the code can expose the core logic of the binary's design and purpose. YETI enables researchers to explore the idea that features derived from a binary's code add value when trying to solve problems in this space, which was not easily possible before it was developed. While the additional processing required to generate these features adds challenges in scalability, results indicate that this approach may both be feasible computationally and add accuracy and flexibility when building analytics and machine learning models for binary files.

[Photo credit: iStock.com/undefined undefined]

How does YETI extract code features?

To gain a deeper understanding of binary files, YETI has leveraged the open-source reverse-engineering tool Ghidra (developed and maintained by NSA's Computer and Analytic Sciences Research Group). Ghidra is able to both disassemble and decompile the machine code of a binary so that the functionality can be analyzed and understood. In addition, Ghidra also provides a rich scripting application programming interface (API) and the ability to run scripts "headlessly," meaning that Ghidra can run a script on a file or set of files and extract data of interest for later analysis. By integrating Ghidra, YETI is able to gain access to a rich array of new features that are now being explored more deeply.

Some analytics built in YETI are focused on features already identified by Ghidra, for example immediate values used in the disassembly, cross-references of functions, or x86/64 instructions. Other analytics can use these features as building blocks for more complex analysis, opening doors to possibilities that are limited only by the imagination of the researcher creating them.

Details of YETI's architecture

YETI is built on top of a variety of technologies. The user portal for YETI is a web application developed in Plotly Dash, which is a useful web framework for researchers because it abstracts many of the design and interface details, supports the interactive charts that many of YETI's analytics leverage, and allows development in Python. Elasticsearch is used to store extracted data. Each component of YETI, for example the Elasticsearch instances, the web application, and the analytic workers, is run inside a Docker container. Celery is used for scheduling tasks such as running an analytic and delivering the resulting data to the component that consumes it. Kubernetes is used to orchestrate the various components needed to run YETI; it provides many benefits such as load distribution, recovery from some kinds of failures, and the ability to scale up or down resources to complete analytic tasks.

To enable development of analytics, YETI also has JupyterLab integrated into its architecture as a first-class citizen, meaning it can access Elasticsearch to retrieve and store data, schedule Celery tasks to run workers and collect data, and other features needed to prototype and test a new analytic. This allows researchers to experiment with different possibilities and build a capability they feel provides value before they build it into YETI as part of the user interface, which requires more of a time investment and is more cumbersome to test. See figure 1 for a simplified diagram of the YETI architecture.

Challenges of scaling YETI's analytics

Scalability is still a largely unexplored question in this space. Tools exist with similar architectures (e.g., built on top of Kubernetes, Docker, and Elasticsearch, doing analysis of binary files) that have reportedly been successfully scaled to handle millions of files. YETI tends to focus on analytics that process and analyze disassembly or decompilation in various ways, which is quite computationally intensive. So far, YETI has been used on much smaller populations of files (in the thousands). Similarly, in the machine learning that YETI applies, there are challenges with applying the algorithms over the quantities of data generated by YETI's analytics. There are a number of strategies which may help in stretching capabilities further, such as using free compute cycles to pre-compute and store results, optimizing data storage formats for the type of analytic the data is being used for, and filtering the data into smaller subsets based on some pre-analysis to reduce the number of files an analytic is run over.

Use cases for YETI

YETI has been used to develop a wide variety of analytics to date, ranging from workers that extract features of code and produce charts based on them, to machine learning capabilities that consume features generated by analytics and attempt to find relationships between files or groups of files. Typically when a new feature is being explored, the data of interest is extracted from a set of files and exploratory data analysis is conducted to determine meaningful patterns and inform decisions on how to display it, or how to apply machine learning algorithms to it.

In this section, several analytics that have been explored within YETI are discussed, along with results. YETI was used to curate the data in these experiments and to extract features explored in these analyses. Some of the most promising results from the work are described below. Better understanding of the value of function prologue data through

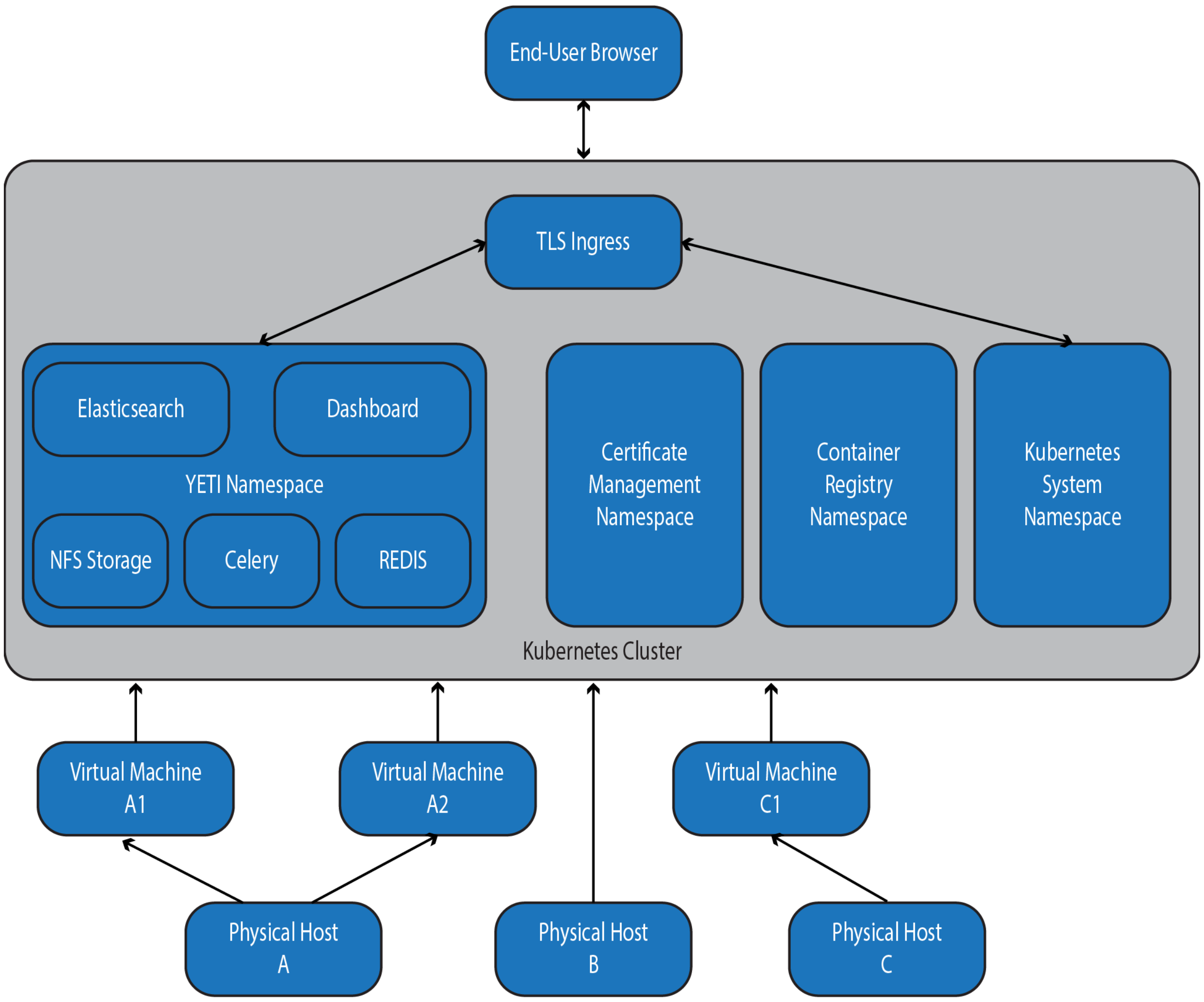


FIGURE 1. This diagram shows a simplified version of the YETI architecture.

exploration in YETI is an area of active research within the team.

Function prologues

Binaries contain compiled code, and code tends to be organized into functions, or pieces of code that perform specific tasks that can be re-used, or "called," as needed. When a program is compiled, the compiler tends to have certain patterns of assembly instructions that are used at the beginning and end of each function; these patterns handle setting up the stack and registers with the expected data and are generally specific to both the compiler and the architecture that the program is being compiled for. This makes them an interesting feature to use in assessing if two files are similar to one another; similar function prologues may mean that the code was compiled with the same compiler and options, and for the same architecture. To test this theory, several summer interns working with the YETI team explored using function prologues to help categorize files using machine learning.

A Ghidra script was developed to extract the first few instructions from each function in a binary. Because the length of a prologue is unpredictable, part of the analysis involved experimenting with various lengths and trying to find the ideal sets of instructions to consider as prologues. The amount of input was then reduced by looking across functions

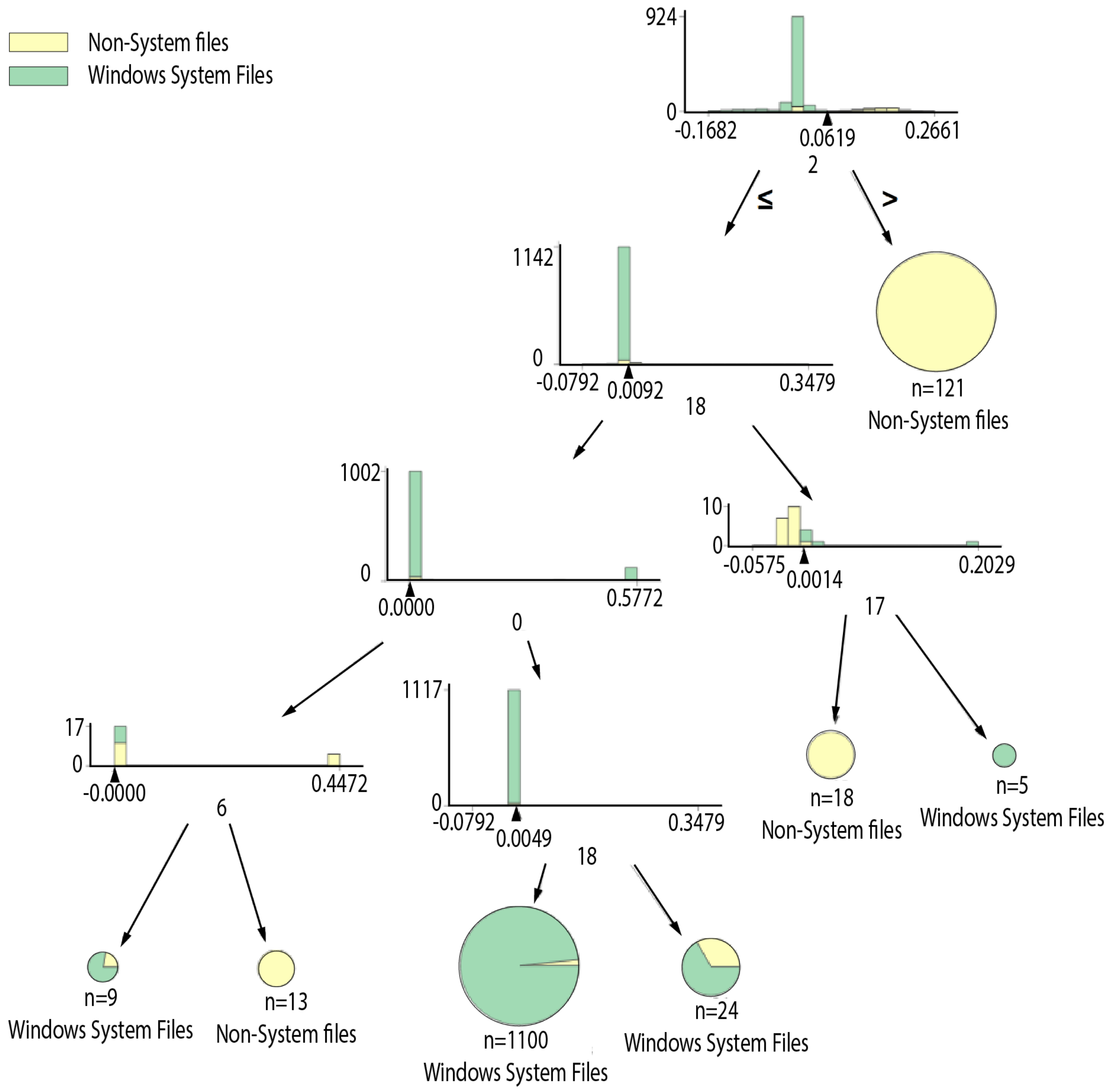


FIGURE 2. This decision tree shows classification decisions for a collection of files.

and files to determine the most common sets of prologue instructions. For this work, a body of 2,500 binaries produced 1.5 million prologue features; singular value decomposition was used to reduce this to 20 features.

Using this data, a random forest model was trained and used for two-class classification between system and non-system binary files. It was able to detect 99.6 percent of system files correctly, and 76 percent of non-system files.

To better understand the classification process, a second experiment was performed on the same data using decision trees instead of a random forest. It had similar performance, with 99 percent of system files correctly detected, and 72 percent of non-system files. This was done in order to gain insight into how classification decisions were being made by the model. Figure 2 shows an example of one of the visualizations generated.

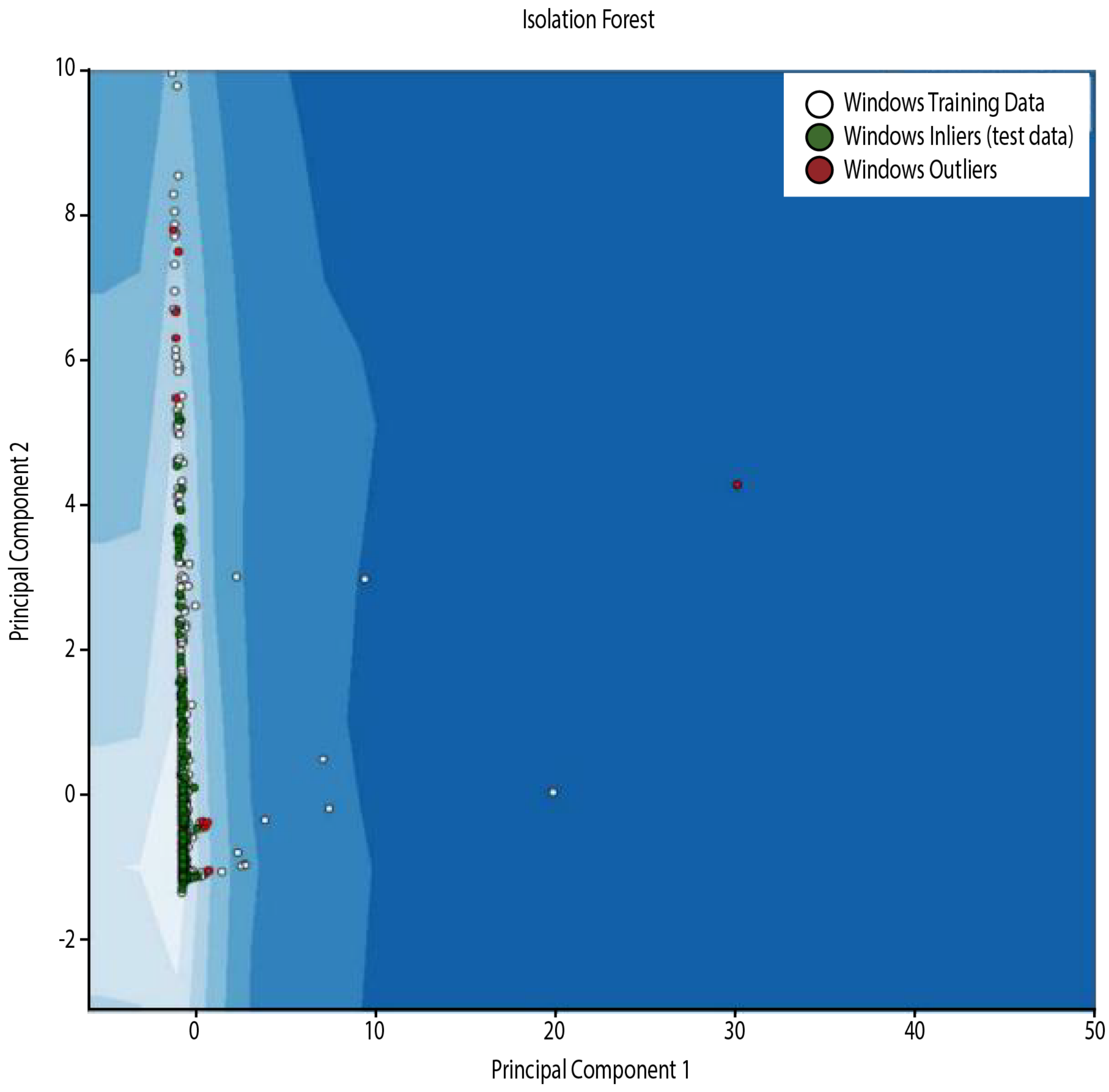


FIGURE 3. These isolation forest results show the Windows system test set for x86 Windows prologues.

In this decision tree, at each node in the graph, in cases where there is a mix of system and non-system files, the distribution of the data is shown. The black carrot under the distribution indicates where the algorithm chose to divide the data. The number from 0-19 below the distribution is the dimension of the data that was used to make this determination. This illustration of the process is useful in gaining an understanding of how the classification process progresses.

Another separate experiment to help measure the analytic value of function prologue data was training and testing an isolation forest, which is an unsupervised learning algorithm that detects anomalies by isolating outliers in the data. The model was trained on Windows files, and figure 3 and 4 compare the results of testing it against additional Windows system files, and non-system files.

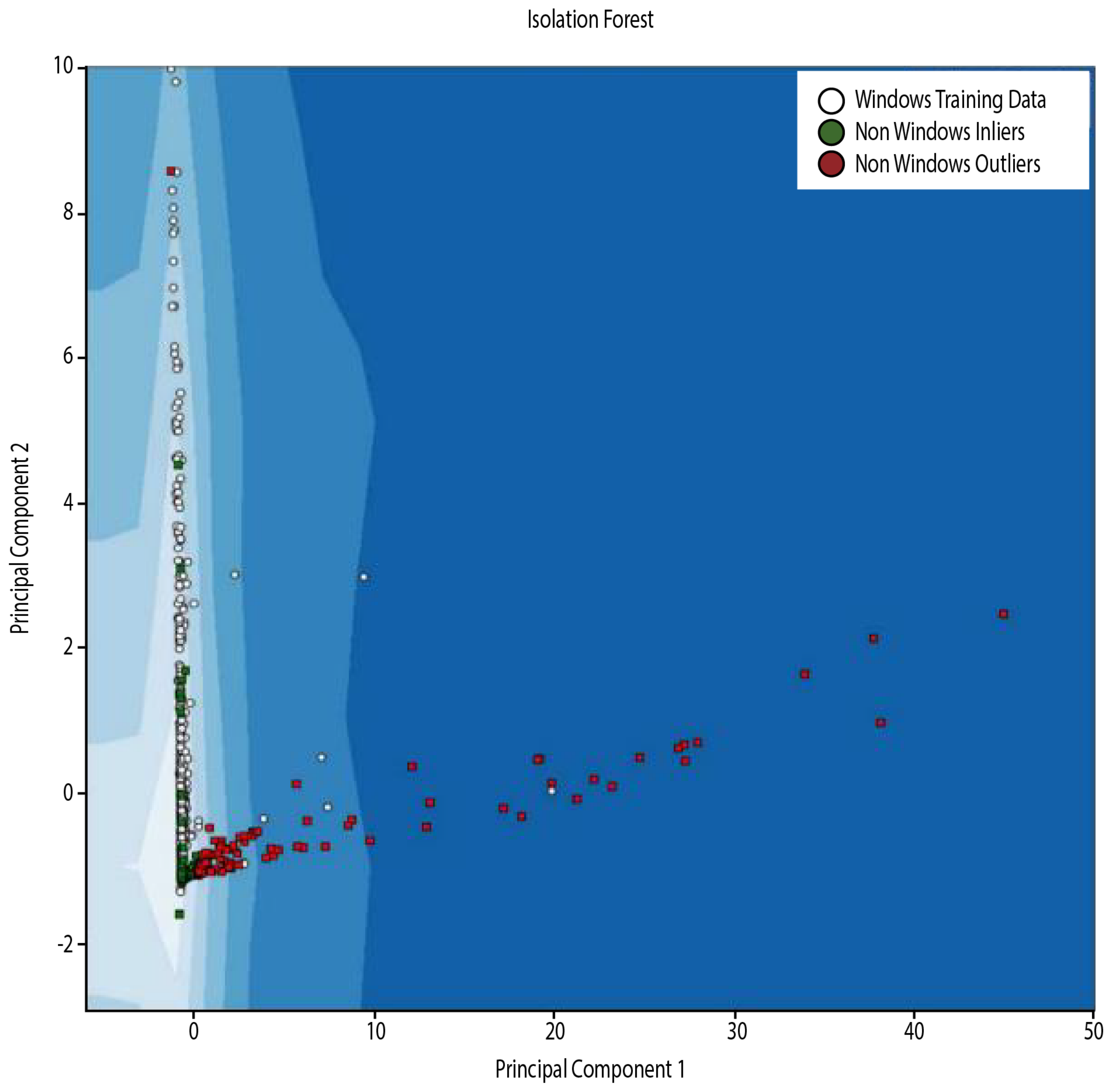


FIGURE 4. These isolation forest results show x86 non-system test set prologues.

To interpret figures 3 and 4, white points are the training set files; green points are the files detected as inliers, and red points are files detected as outliers by the algorithm. Shades of blue indicate the varying outlier scores assigned, with darker blue being a higher outlier score.

Comparing figures 3 and 4, more non-system files were detected as outliers (52 percent) versus Windows system files detected as outliers (5 percent). Because the percent detected as outliers varies depending on the contamination parameter used with the model, the point of interest is the difference between the percentages for each class of files. We can infer from this large difference that there are many features of the non-system files that set them apart from the Windows system files.

As a final note on these results, while it may not seem substantial to detect only 52 percent of non-system files as outliers, consider that information was lost during principal component analysis (PCA), and also that this is focused on a single feature; in future work it would be worthwhile to explore performance with additional features included.

Index of coincidence

The YETI team has been performing exploratory data analysis by calculating an index of coincidence (IC) score over the bytes of code in an executable file. Coincidence counting is a technique which places two texts side-by-side and counts the number of times that identical letters appear in the same position in both texts. This count is taken either as the ratio of the total, or normalized by dividing by the expected count for a random source model [1]. The formula for calculating the IC is shown in equation 1. Because in code, like in natural language text, we expect certain values to appear more often than others, the IC is higher than it would be for random values. Furthermore, code generated by different compilers may be able to be characterized by the IC calculation, which would be useful in measuring how similar two binaries are to one another.

$$IC = c\sum_{i=1}^{c}\frac{n_i}{N}\frac{n_i - 1}{N - 1} \qquad (1)$$

To calculate the IC, we wrote a Ghidra script that iterated over the functions in a program and performed the calculations over the bytes in the body of the function. Files that did not contain any functions were removed from the dataset. Each file had one non-negative number representing the IC.

In the experiments measuring IC on code and testing it as a feature, we encountered several challenges. One was the size of the available datasets. To accurately characterize code from different compilers, we theorize that a much larger quantity of data may be needed. In some cases, for specific sets of files, it may be difficult or impossible to find a large enough quantity of available data.

We also observed that better results were achieved with IC when instead of calculating over the entire body of the function, we identified the bytes that were most meaningful to the purpose or label of the file and filtered out bytes that were less impactful for the purpose of the file. We explored several methods to identify these bytes, including:

- Trying to remove the prologue and epilogue bytes from the calculations, and
- Using entropy and cross-entropy calculations on different sections of the function to try to identify changes that might indicate where the substance of the functions was.

Finally, we found that experiments training models on IC calculations had poor classification results when used individually, but when combined with function prologue data, the IC was consistently within the top 10 percent of most important features of the classifiers. This demonstrates that while by itself IC is not distinguishing, at least for the amount of data available, at later points of the classification process, IC may become better able to divide the data into effective groupings.

Entropy and cross-entropy

In a similar vein to the IC, another analytic we developed involved calculating entropy over the distribution of the raw bytes of a file. Entropy calculations contain information about the distribution and counts of bytes that make up a file; see equation 2 for the formula used to calculate it in YETI's experiments. The same strategy used with the IC calculations proved most effective, namely extracting the most relevant bytes from the functions, and eliminating files without functions from the dataset.

$$H(X) = \sum_{i=1}^{c} -p(x_i)log_2p(x_i) \qquad (2)$$

Entropy has been successfully used in previous research on malware detection, which was one reason we chose it for further exploration [2, 3].

Cross-entropy is a similar metric that serves to allow comparison between two probability distributions, with a higher value for the cross-entropy calculation indicating that the two distributions are further apart from one another [4]. To explore this feature, we initially calculated each file's cross-entropy with every other file in the dataset (see equation 3). Using this data, for some categories where files had sufficiently similar scores, we identified a representative file so that cross-entropy calculations did not have to be stored for *every* file.

$$H(p,q) = \sum_{i=1} -p(x_i)log_2q(x_i) \qquad (3)$$

Results showed that cross-entropy was able to differentiate between system and non-system files with around 90 percent accuracy in each case.

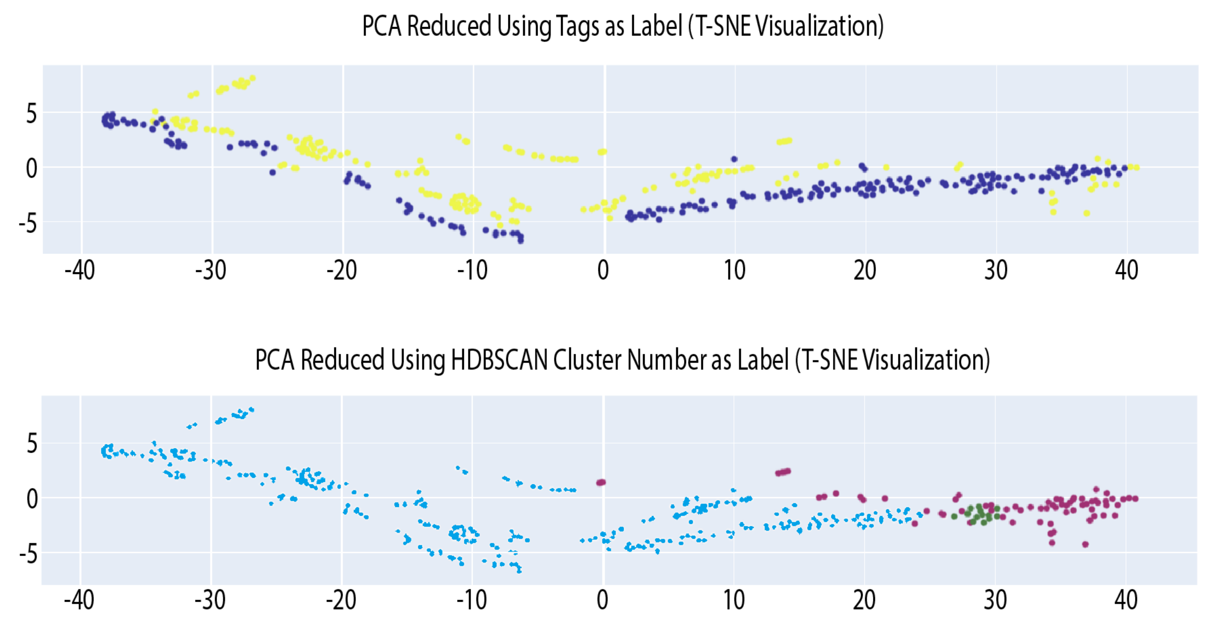


FIGURE 5. This attempt at clustering did not succeed at breaking data into desired clusters.

Clustering binaries

YETI has also been used for research into clustering binary files, building off of existing research on clustering malware [5]. The aim is to see which features contribute positively to clustering binary data. Once "good" clusters have been established, the next step is to look at the distribution of the features based on the clusters and determine what is considered normal for that cluster (i.e., that subset of binary files). Many of the features tested with the clustering algorithms, such as function cross-reference counts, are extracted using YETI to run Ghidra scripts on the binaries.

In figure 5, PCA is used to reduce the dimensionality of the data. The data includes Windows system files, and a relatively new dataset curated and provided by the NSA Research Directorate's Laboratory for Physical Sciences called Assemblage; features extracted from the data and used for clustering include standardized/one-hot encoded header data, function cross reference counts, and instruction occurrences. We then used hierarchical density-based spatial clustering of applications with noise (HDBSCAN) to cluster and assign labels to each file (including a -1 noisy label, which is the purple color in the lower image in figure 5) [6]. The top image in figure 5 shows the two-dimensional (2-D) projection of the data with the marker color being the known label (Windows or Assemblage). The bottom image shows the 2-D projection of the data with the marker color being the HDBSCAN cluster label. Overall, we can see that most of the points are labeled light blue, with a few points labeled green and a few others labeled the noisy purple.

In figure 6, PCA is again used to reduce the dimensionality of the data, but this time instruction occurrences are excluded as a feature, and HDBSCAN is used to cluster and assign labels to each file. Once again, the top image shows the 2-D projection of the data with the marker color being the known label (Windows or Assemblage). The bottom shows the 2-D projection of the data with the marker color being the HDBSCAN cluster label. In this case, there is better delineation between the two known file families. Most of the blue points in the top chart are clustered together in the bottom. Some of the yellow points are properly clustered together (the top darker pink cluster) while the rest are broken down into further clusters or marked as noisy.

There are many applications for successful clustering of binary files. Within YETI, one goal is to be able to automatically tag files with relevant labels based on clusters they fall into rather than relying on users

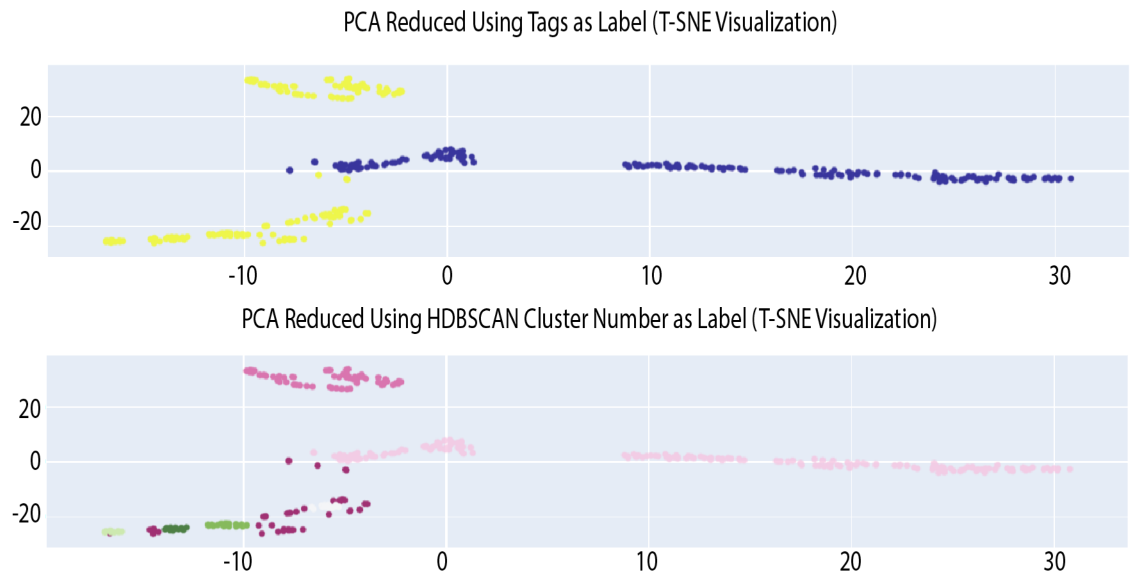


FIGURE 6. We achieved this more successful attempt at clustering by removing noisy features.

to correctly tag data upon upload. A more common use case is of course being able to identify suspicious files if they categorize into clusters with known malware, or to gain insights into attribution of a file back to its author [7].

Conclusion

YETI is a flexible and robust tool for research on scalable binary analysis to better understand binary similarity and quantify relationships between binaries in meaningful ways. There is a great deal of ongoing and planned work exploring the features and analytics described above, as well as developing new analytics for use in YETI. YETI houses a growing set of useful metrics for code similarity within a framework that allows experimentation and analysis to generate innovative tools and techniques in this space. By tapping into the ability to identify and understand the code within binaries, new doors are opened to leverage features that convey detailed information about the purpose and origin of binary files, as well as their relationships to other binaries. This potential for deeper understanding may represent the future of where binary analysis is headed. The YETI team is excited about both the system engineering challenges of scaling the analysis, as well as the research being done in identifying new meaningful features of binaries and applying them within machine learning frameworks to solve hard problems.

References

[1] Friedman W. *The Index of Coincidence and its Applications in Cryptography.* Department of Ciphers Publ 22. Geneva (IL): Riverbank Laboratories; 1922.

[2] Gilbert D, Mateu C, Planes J, Vicens R. "Classification of malware by using structural entropy on convolutional neural networks." *Proceedings of the AAAI Conference on Artificial Intelligence.* 2018;32(1). Available at: https://doi.org/10.1609/aaai.v32i1.11409.

[3] Lyda R, Hamrock J. "Using entropy analysis to find encrypted and packed malware." I*EEE Security and Privacy.* 2007;5(2):40–45. doi: 10.1109/MSP.2007.48.

[4] Cybenko G, O'Leary DP, Rissanen J, editors. *The Mathematics of Information Coding, Extraction and Distribution.* New York (NY): Springer; 1999. ISBN: 978-0-387-98665-4.

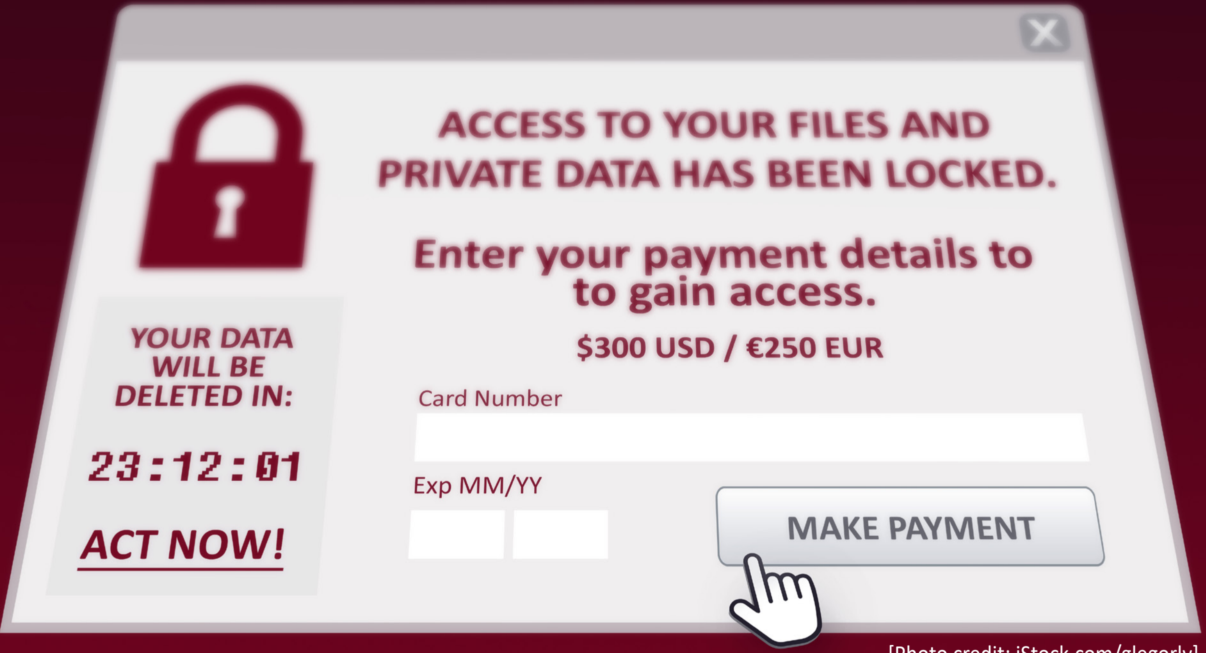
[5] Kim S. "PE header analysis for malware detection." (2018). Master's Projects, 624, San Jose State University. Available at: https://doi.org/10.31979/etd.q3dd-gp9u and https://scholarworks.sjsu.edu/etd_projects/624.

[6] McInnes L, Healy J. "Accelerated hierarchical density based clustering." In: *2017 IEEE International Conference on Data Mining Workshops (ICDMW);* 2017; IEEE: pp 33–42. Available at: https://doi.org/10.1109/ICDMW.2017.12.

[7] Rosenblum N, Zhu X, Miller BP. (2011). "Who wrote this code? Identifying the authors of program binaries." In: C*omputer Security-ESORICS 2011: 16th European Symposium on Research in Computer Security;* 2011 Sep 12–14; Leuven, Belgium: pp. 172–189). Springer Berlin Heidelberg. Available at: https://doi.org/10.1007/978-3-642-23822-2_10.

Stopping the Spread Before it Starts

NSA's Office of Research and Technology Applications (ORTA)


ACCESS TO YOUR FILES AND PRIVATE DATA HAS BEEN LOCKED.
Enter your payment details to to gain access.
$300 USD / €250 EUR
Card Number
Exp MM/YY
MAKE PAYMENT
YOUR DATA WILL BE DELETED IN:
23:12:01
ACT NOW!

[Photo credit: iStock.com/glegorly]

Two NSA Inventors Solve the Problem with Ransomware

On the heels of the successful commercialization of AutoBerry technology [1], NSA innovators Daryle Deloatch and Mark Haney were ready to look for a new big problem to solve. "We wanted to anticipate the next big issue and address it," said Deloatch.

Enter ransomware: this mechanism of exploitation has been a major cybersecurity threat for years, thanks to the explosion of the use of digital devices.

The coronavirus pandemic uncovered a wealth of vulnerabilities since the world suddenly had to operate remotely. Ransomware attacks skyrocketed: IEEE's *IT Professional* states that ransomware attacks increased by 150 to 200 percent during the pandemic [2]. It was clear to Deloatch and Haney that this was a weakness that needed to be addressed.

The pair began by looking into current solutions to ransomware attacks. What was stopping them from effectively minimizing the impact of an attack? Deloatch explained that current mitigations to combat ransomware are reactive and are often deployed too late to prevent any damage from being done. "Many of the mitigations out there are flawed. They use a centralized scanner, which is a single point of failure," he said. "Or, the scanner is located in the wrong place when the ransomware is launched, which—at that point—is too late to stop it from spreading."

Their answer: create a solution that can search multiple areas of a network at once to speed up the inspection, and ultimately, neutralization process. By deploying agents across a system, their technology is able to provide a system-wide view, allowing for detection of malware that would usually go unnoticed by a single centralized scanner. Since ransomware attacks usually perform some preemptive actions before launching (i.e., shutting off ports, deactivating security processes, etc.), the multiple area, system-wide scans make it easy to identify vulnerable situations that could set the stage for a possible attack.

The entire process—which can be completed in nanoseconds—can stop the spread of different types of ransomware like password lockouts and encryption or security changes. One key benefit of the technology is its ability to run multiple, continuous scans without taxing the performance of the system. The proactive design and small footprint enable it to be used in various applications including smartphones, drones, Internet of Things devices, Supervisory Control and Data Acquisition systems, and cloud storage environments.

After the technology, titled "Security system for hardening a digital system against malware and method of operation," was submitted to the US Patent and Trademark Office, NSA's Office of Research and Technology Applications (ORTA) pitched a licensing opportunity to Kapalya, a cybersecurity startup that previously licensed another NSA encryption technology. The company licensed the technology and worked with Deloatch and Haney to create a prototype, which exceeded company expectations.

Kapalya has since filed for an exclusive license and recently won a Small Business Innovation Research grant from the National Science Foundation for phase II funding. The funding will allow Kapalya to complete further research and development; the company plans to incorporate artificial intelligence to provide additional security based on learned behavioral activity.

Deloatch and Haney plan to keep creating as well—the pair is looking to submit additional patents to supplement their ransomware technology until they come across their next big problem to solve.

References

[1] NSA. "From Fort Meade to the marketplace: Successes in technology transfer." *The Next Wave.* 2012;19(3):14–17.

[2] Baig ZA, Mekala SH, Zeadally S. "Ransomware attacks of the COVID-19 pandemic: Novel strains, victims, and threat actors." *IT Professional.* 2023;25(5):37–44. Available at: https://doi.org/10.1109/mitp.2023.3297085.

Featured Publications by NSA Researchers, 2022–2023

Bader DA, Burkhardt P. "A simple and efficient algorithm for finding minimum spanning tree replacement edges." *Journal of Graph Algorithms and Applications.* 2022;26(4):577-588. Available at: https://dx.doi.org/10.7155/jgaa.00609.

Comar B. "Modulation recognition using YOLOv5 on the WBSig53 dataset." In: *IEEE Wireless and Optical Communications Conference (WOCC),* 2023, pp.1-6. Available at: https://ieeexplore.ieee.org/document/10139594.

Comar B. "Using Kullback Leibler divergence to perform modulation recognition on a TorchSig dataset." In: *IEEE International Conference on Communication Systems and Network Technologies (CSNT),* 2023, pp. 470-477. Available at: https://ieeexplore.ieee.org/document/10134740.

Comar B. "Using mutual information to perform modulation recognition on the Sig53 dataset." In: *IEEE Wireless and Optical Communications Conference (WOCC),* 2023, pp. 1-6. Available at: https://ieeexplore.ieee.org/document/10139391.

Careers with NATIONAL IMPACT

Work where continual learning, work-life balance and contributing to the greater good are top priorities.

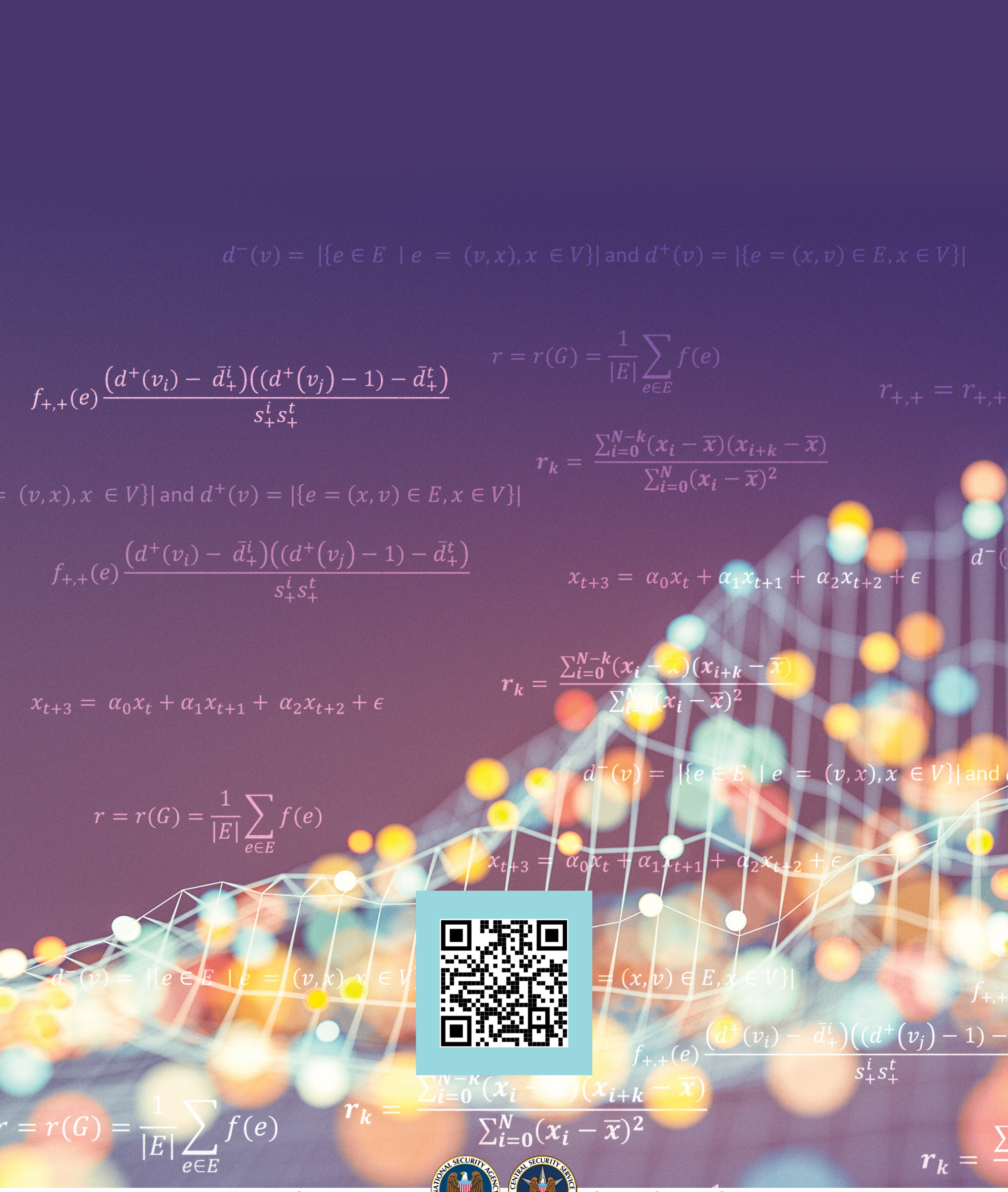
NSA keeps the nation safe through our signals intelligence and cybersecurity missions. If you have a heart for service, there's a place for you here.

Apply now at
IntelligenceCareers.gov/NSA

Business & Contracting
Cybersecurity
Foreign Language
Intelligence
STEM
Professional Support

NSA Careers

U.S. citizenship required. NSA is an Equal Opportunity Employer.


$d^-(v) = |\{e \in E \mid e = (v,x), x \in V\}|$ and $d^+(v) = |\{e = (x,v) \in E, x \in V\}|$
$f_{+,+}(e) \frac{(d^+(v_i) - \bar{d}^i_+)((d^+(v_j) - 1) - \bar{d}^t_+)}{s^i_+ s^t_+}$
$r = r(G) = \frac{1}{|E|} \sum_{e \in E} f(e)$
$r_k = \frac{\sum_{i=0}^{N-k} (x_i - \bar{x})(x_{i+k} - \bar{x})}{\sum_{i=0}^{N} (x_i - \bar{x})^2}$
$x_{t+3} = \alpha_0 x_t + \alpha_1 x_{t+1} + \alpha_2 x_{t+2} + \epsilon$

National Security Agency

Central Security Service

Defending Our Nation. Securing The Future.